
# OS PORTING AND ABSTRACTION LAB USER MANUAL

**Release 1.3.8**


MapuSoft Technologies
1301 Azalea Road
Mobile, AL 36693
www.mapusoft.com

# Table of Contents

# Chapter 1. Introduction to OS PAL

This chapter contains the following topics:

# About OS PAL

OS PAL is an Eclipse based IDE. It integrates OS Changer and OS Abstractor with Eclipse's CDT to offer an IDE for developing and porting applications on host platforms.

OS PAL features include:

- Creation of C and C++ OS PAL projects
- Automatic configuration of any OS Changer and OS Abstractor APIs needed by the application
- Custom configuration of OS resources needed by the application
- Custom configuration of OS Abstractor Resources
- Custom configuration of OS Abstractor for single or multi-application development (Process Feature support)
- Create Ada Projects
- Host simulation
- Full Source Library Package generation
- Convert Ada source code to C/C++ code
- Compile Ada code to an executable
- Do Platform and Application profiling

OS PAL uses OS Abstractor and OS Changer technology to produce optimized target code:

- Up to 9 target configuration tabs to optimize the target code specific for your application
- Generated target code is optimized to contain only the APIs used by the application
- Allows for further optimization by in-lining user selected API's

Contact MapuSoft to receive the components needed for using OS PAL. The steps for using OS PAL are comprehensively described in the following pages. The OS Porting and Abstraction Lab User Manual has screen shots for many of the steps.

# Installing OS PAL

You can download an evaluation copy from our website or install OS PAL via the evaluation CD given by MapuSoft Technologies.

To install OS PAL**:**

1. Insert the CD and run it. A welcome html page will be auto run.
2. Click **Port, abstract and optimize code on a host OS PAL** link.
3. Select the host. For Example: windows or Linux.
4. Click **Click here to begin installation**. This will launch the ospal installer which is called *os-porting-and-abstraction-lab-windows-jvm-installer.exe*.

   **NOTE**: This *.exe* file is what you have, if you download OSPAL from the web manually.
5. Double click on the **.exe** file to launch the installer.
6. You can view the installation system requirements displayed on the installer wizard.
7. Browse and select the installation directory.

   **NOTE**: By default, it is c:\MapuSoft\OSPAL.Do not install OS PAL in any location with spaces in the path name reference such as any subdirectory of "Program Files" or "My Documents", or the like. OS PAL may have problems with paths containing spaces, and if not, usually other programs used with OS PAL will experience problems with such paths.

# Uninstalling OS PAL

To uninstall OS PAL:

1. Browse to the installed OSPAL directory and start the **Uninstall** application.
2. You can also uninstall OS PAL by selecting **Control Panel> Add/Remove Programs.** Select OS PAL and click **Remove**.
3. There is a possibility of user generated/modified files to be left on your **OSPAL** installation directory. If not necessary, delete the files manually to remove the files.

# List of Technical Documentation

Reference manuals can be provided under NDA. Click http:mapusoft.com/contact/ to request for a reference manual.

The document description table lists MapuSoft Technologies manuals.
Using these documents you can:

- Port applications
- Perform OS Abstraction
- Generate optimized code
- Generate Full Library Package OS Abstractor/OS Changer Packages
- Enable profiling

**Document Description Table**

| User Guides | Description |
|---|---|
| System Configuration Guide | Provides detailed description on the system configuration to work with MapuSoft products. This guide:<br>• Describes the system requirements and configurations to get started with MapuSoft Technologies products |
| OS Porting and Abstraction Guide | Provides detailed description of how to do porting and abstraction using OS PAL. This guide:<br>• Explains how to port applications<br>• Explains how to import legacy applications<br>• Explains how to do code optimization<br>• Explains how to generate library packages<br>• Explains on Application profiling and platform profiling |
| Cross-OS Interface Reference Manual | Provides detailed description of how to use OS Abstraction. This guide:<br>• Explains how to develop code independent of the underlying OS<br>• Explains how to make your software easily support multiple OS platforms |
| VxWorks Interface Reference Manual | Provides detailed description of how to get started with VxWorks interface support that MapuSoft provides. This guide:<br>• Explains how to use VxWorks interface, port applications |
| micro-ITRON Interface Reference Manual | Provides detailed description of how to get started with micro-ITRON interface support that MapuSoft provides. This guide:<br>• Explains how to use micro-ITRON interface, port applications |
| pSOS Interface Reference Manual | Provides detailed description of how to get started with pSOS interface support that MapuSoft provides. This guide:<br>• Explains how to use pSOS interface, port applications |
| pSOS Classic Interface Reference Manual | Provides detailed description of how to get started with pSOS Classic interface support that MapuSoft provides. This guide<br>• Explains how to use pSOS Classic interface, port applications |
| Nucleus Interface Reference Manual | Provides detailed description of how to get started with Nucleus interface support that MapuSoft provides. This guide:<br>• Explains how to use Nucleus interface, port applications |
| POSIX Interface Reference | Provides detailed description of how to get started with |

| User Guides | Description |
|---|---|
| Manual | POSIX interface support that MapuSoft provides. This guide:<br>• Explains how to use POSIX interface, port applications |
| Windows Interface Reference Manual | Provides detailed description of how to get started with Windows interface support that MapuSoft provides. This guide:<br>• Explains how to use Windows interface, port applications |
| Release Notes | Provides the updated release information about MapuSoft Technologies new products and features for the latest release. This document:<br>• Gives detailed information of the new products<br>• Gives detailed information of the new features added into this release and their limitations, if required |

# Supported Host Platforms

OS Pal supports the following host platforms:

- Windows XP/7
- Linux
- Solaris*
  *Available soon

Supported Development APIs:

- Cross-OS Interface
- POSIX Interface
- VxWorks Interface
- pSOS Interface
- Nucleus Interface
- micro-ITRON Interface
- Windows Interface

For a list of OS PAL supported target operating systems, click here:
http://mapusoft.com/products/offerings/

# Getting a License for OS PAL

OS PAL is licensed by the following host and target licenses. A 30 day advanced evaluation license is available for the host licenses.
Click http://mapusoft.com/downloads/ospal-evaluation/ to request an evaluation license.

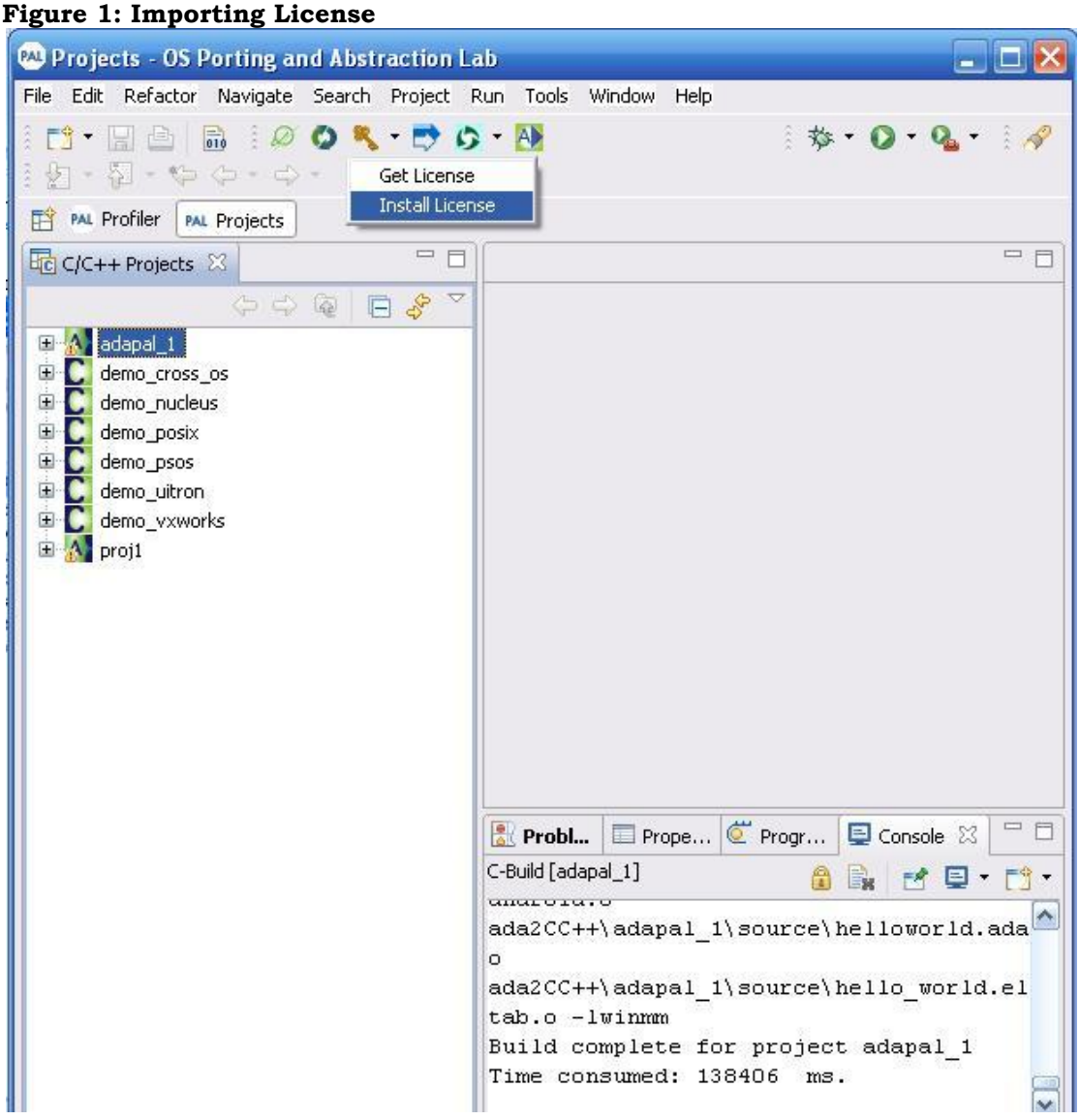
# Installing License for OS PAL

MapuSoft provides a license key to the customers. Once the customers provide the Mac Address of their system, MapuSoft Technologies provides a License key for that particular system. This establishes security for the license.

To install the license:

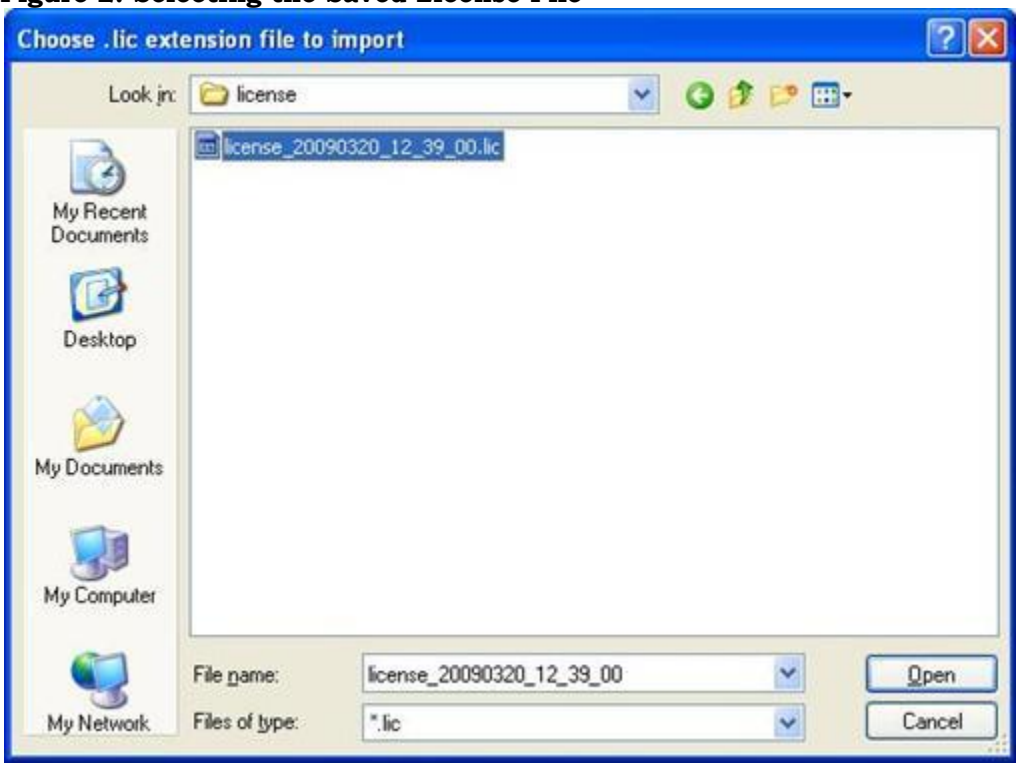
1. Save the license file given to you by MapuSoft.
2. On OS PAL main menu, click the down arrow next to **Key** button and select **Install License** as shown in Figure 1.

**Figure 1: Importing License**



3. Browse to the location of the saved license file and click **Open** as shown in Figure 2. The license key is installed and now you can work on OS PAL.

**Figure 2: Selecting the Saved License File**



# Updating OS PAL

## Getting Updates for OS PAL

**NOTE**: This feature requires OSPAL Host License. Click http://www.mapusoft.com/contact/ to send a request to receive licenses and documentation.

You can get latest OS PAL updates from http://www.mapusoft.com using the following two options:

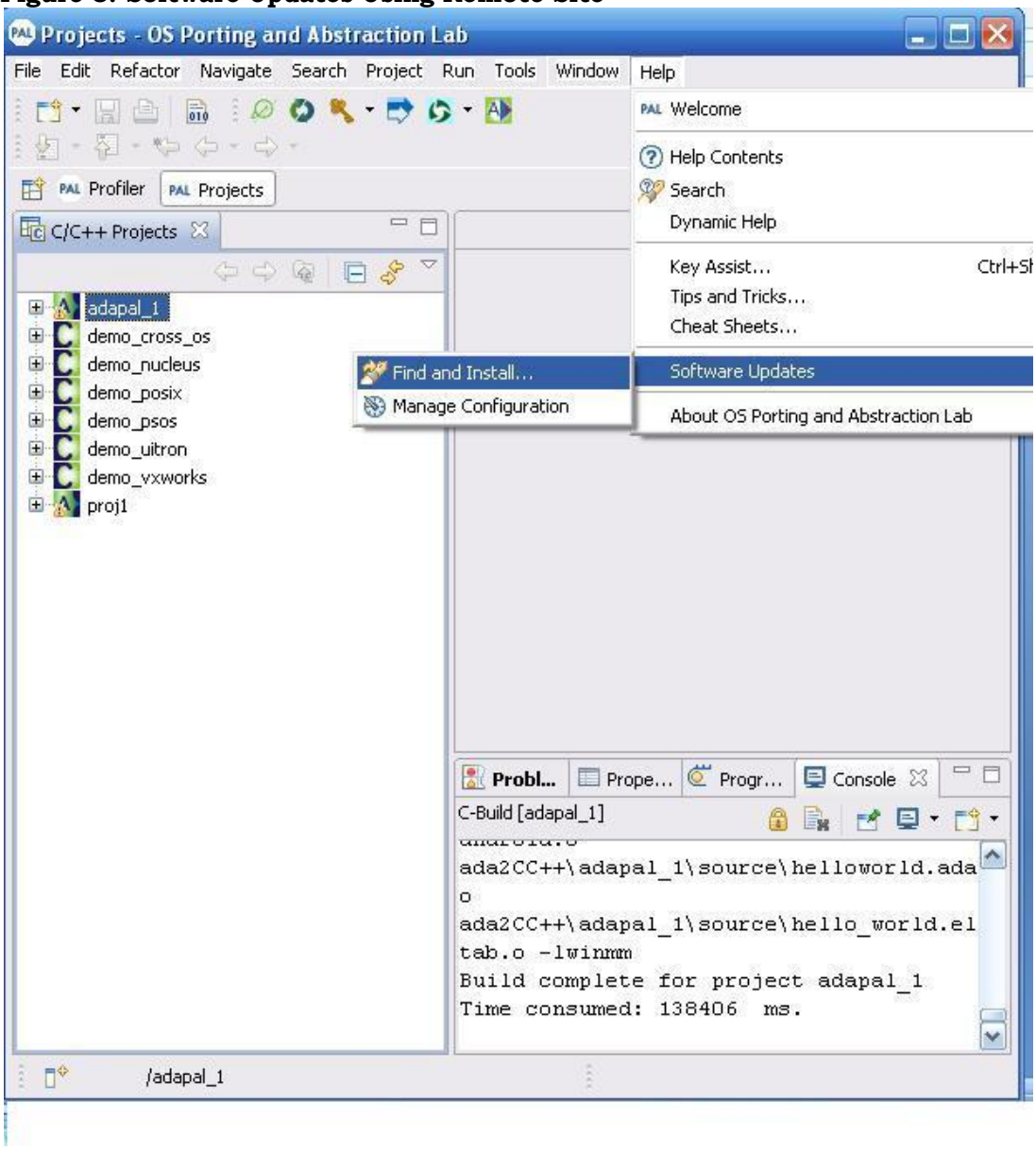
- **Remote Update**: By using Remote Update Site, the system will automatically contact http://www.mapusoft.com/ website and search for the latest updates. You need internet connectivity for this to work.
- **Local Update**: By using Local Update Site, you can do OSPAL updates without connecting to the Internet. For this to work, you need to get the updated files from http://www.mapusoft.com/ by e-mail or CD.

## Updating Software Using Remote Update Site

To update software using Remote update site:

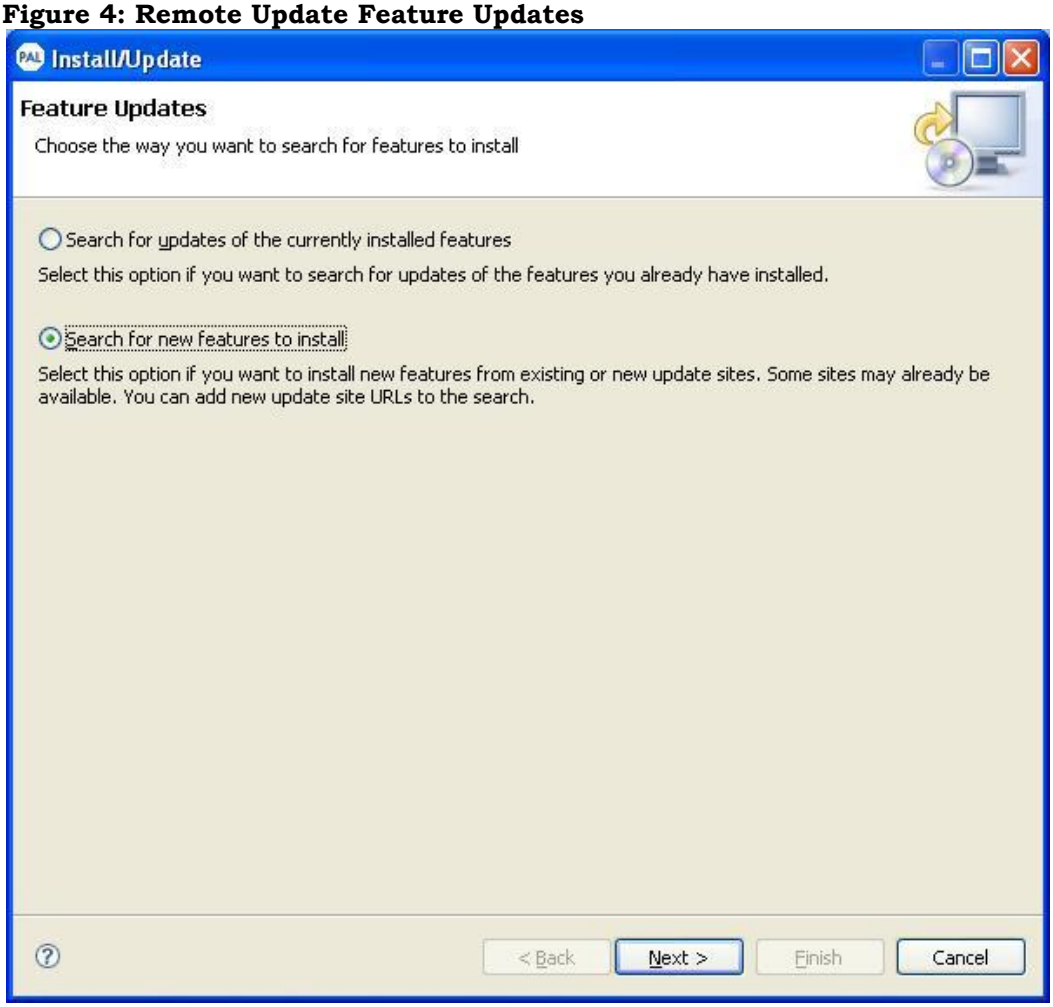
1. From OS PAL main menu, select **Help > Software Updates > Find and Install** as shown in Figure 3.

**Figure 3: Software Updates Using Remote Site**



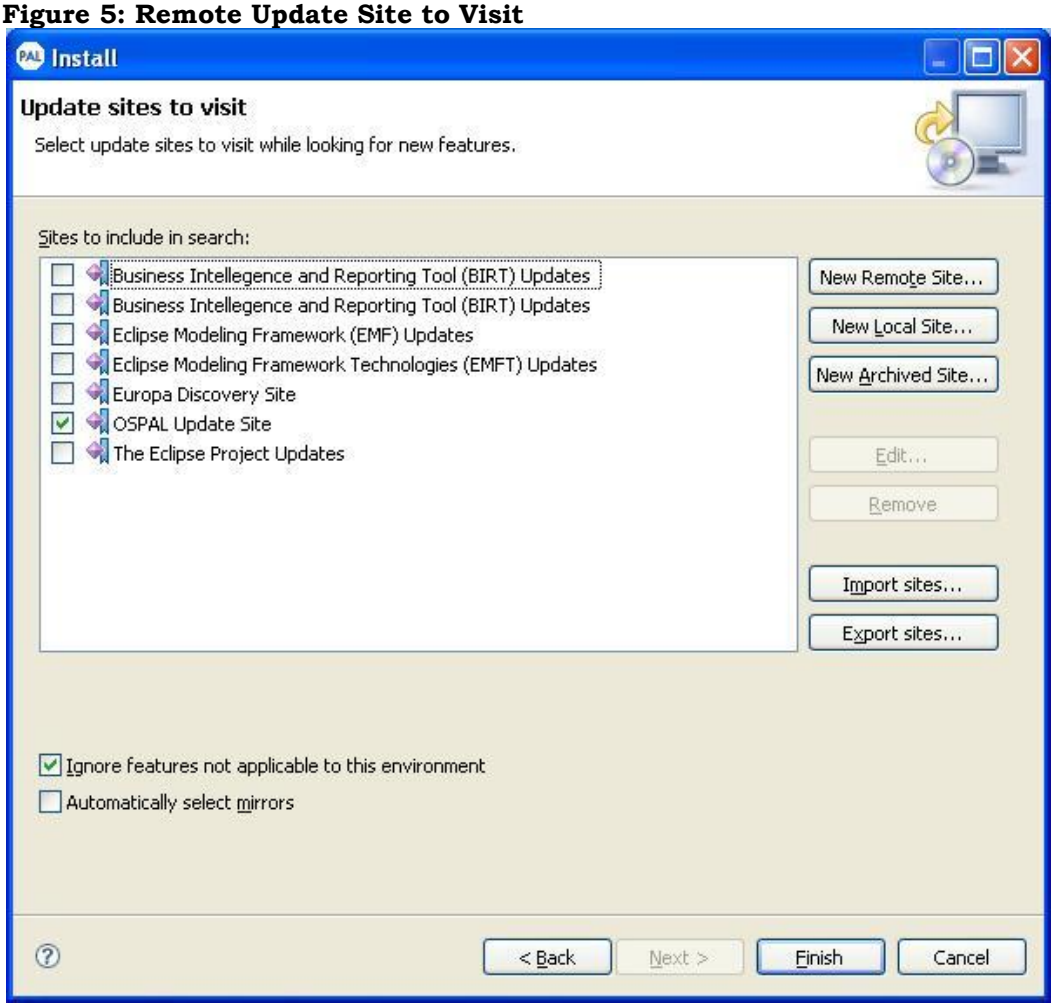
2. Select **Search for new features to install** and click **Next** as shown in Figure 4:

**Figure 4: Remote Update Feature Updates**



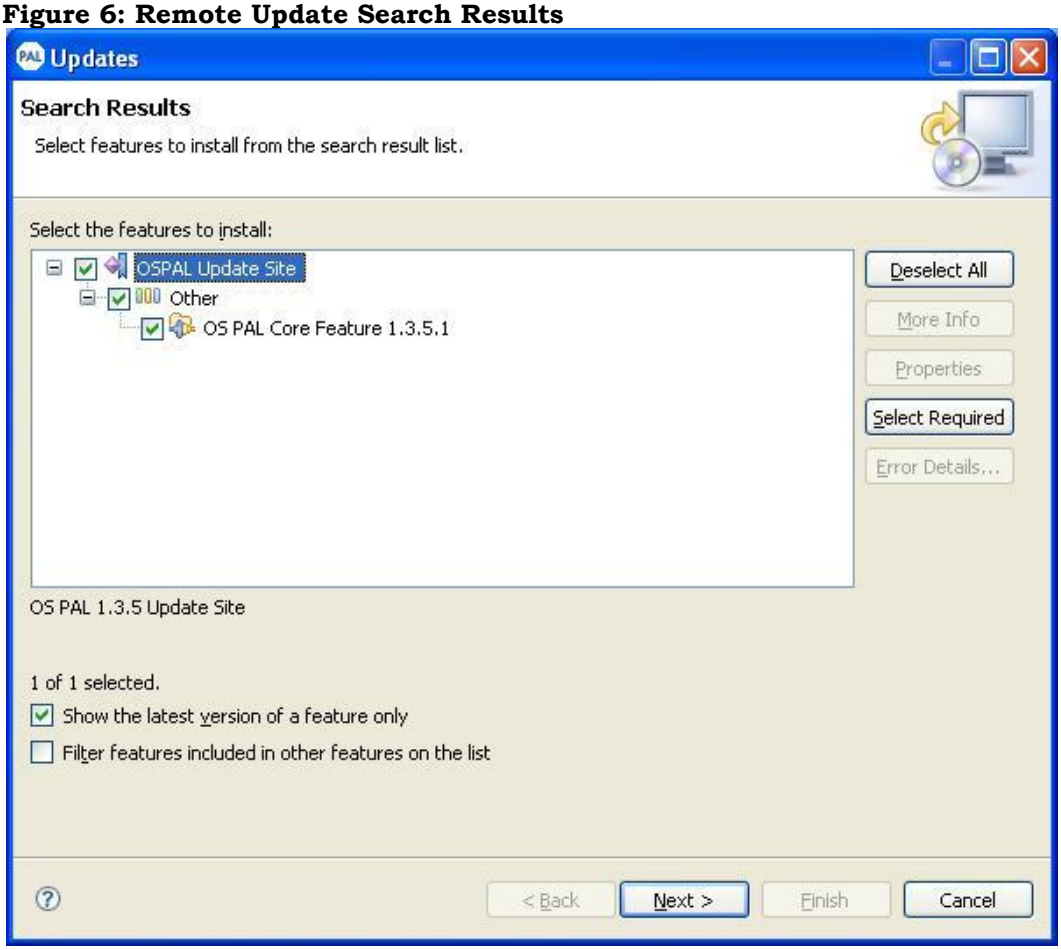
3. On Updates sites to visit window, select the check box next to **OS PAL Update Site** and click **Finish** as shown in Figure 5:.

**Figure 5: Remote Update Site to Visit**



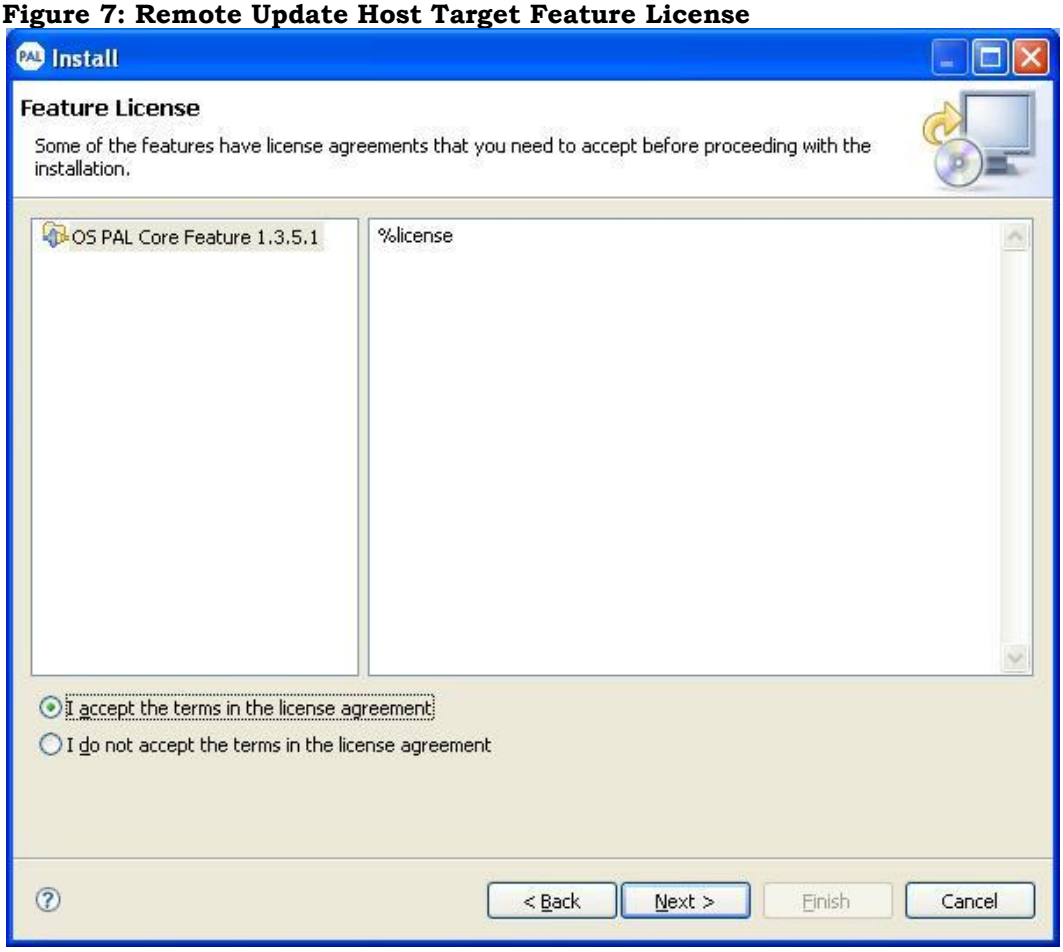
4. On OS PAL Updates Search Results window, select the features under the **OS PAL Update Site** tree parent and click **Next** as shown in Figure 6:.

**Figure 6: Remote Update Search Results**



5. On Feature License window, select the radio button next to **I accept the terms in the license agreements** and click **Next** as shown in Figure 7.

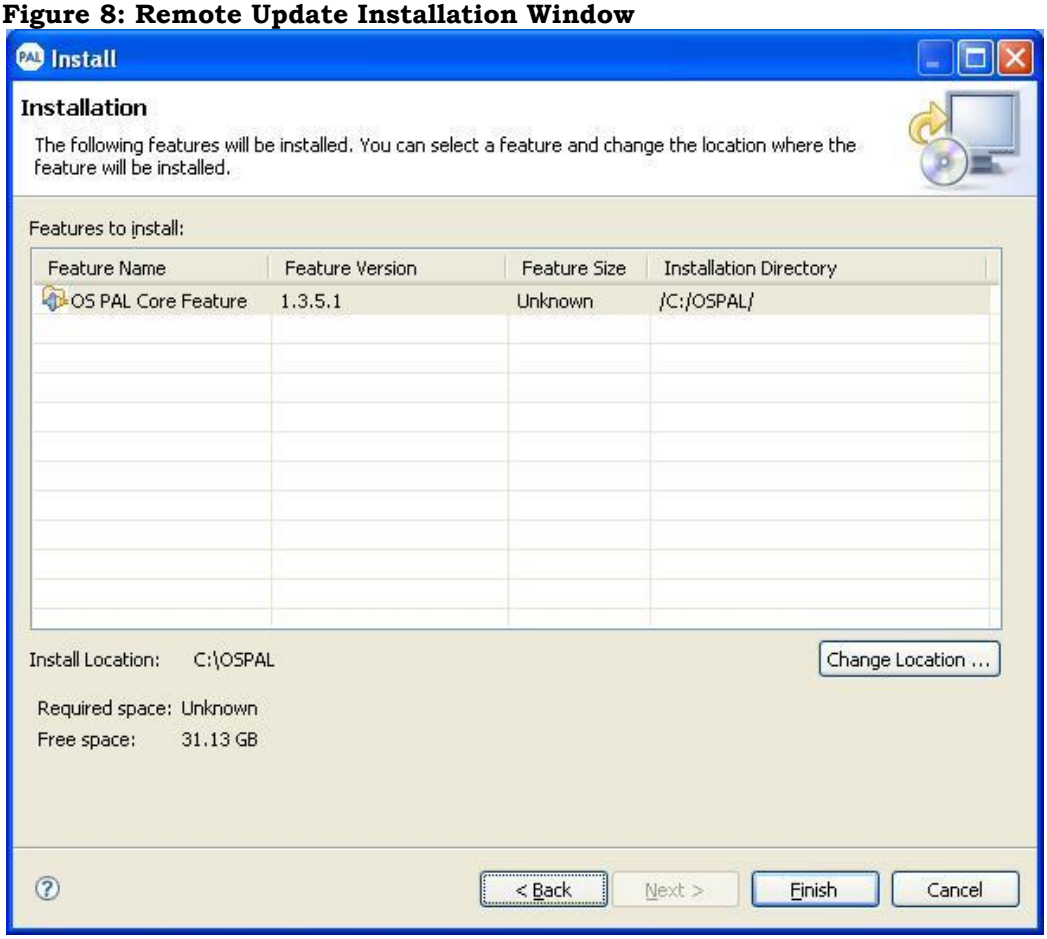
**Figure 7: Remote Update Host Target Feature License**



6. On Installation window, you can view the features that are going to be installed and the Installation Directory as shown in Figure 8.

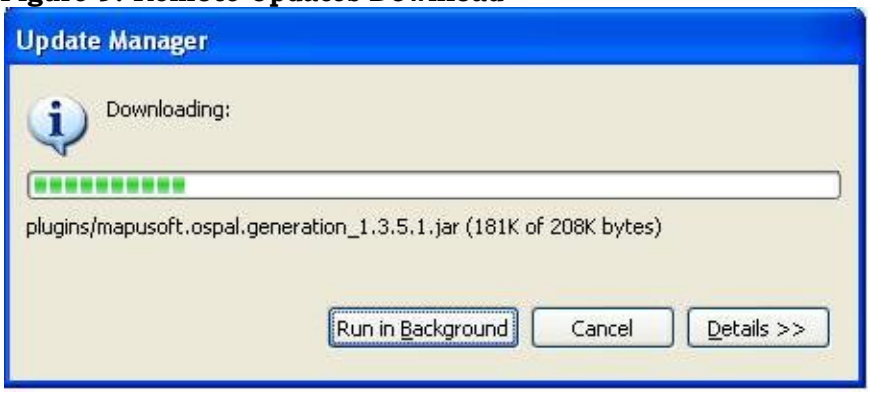
   **NOTE**: You can change the Installation Directory if you want to, by clicking **Change location** and click **Finish**.

**Figure 8: Remote Update Installation Window**



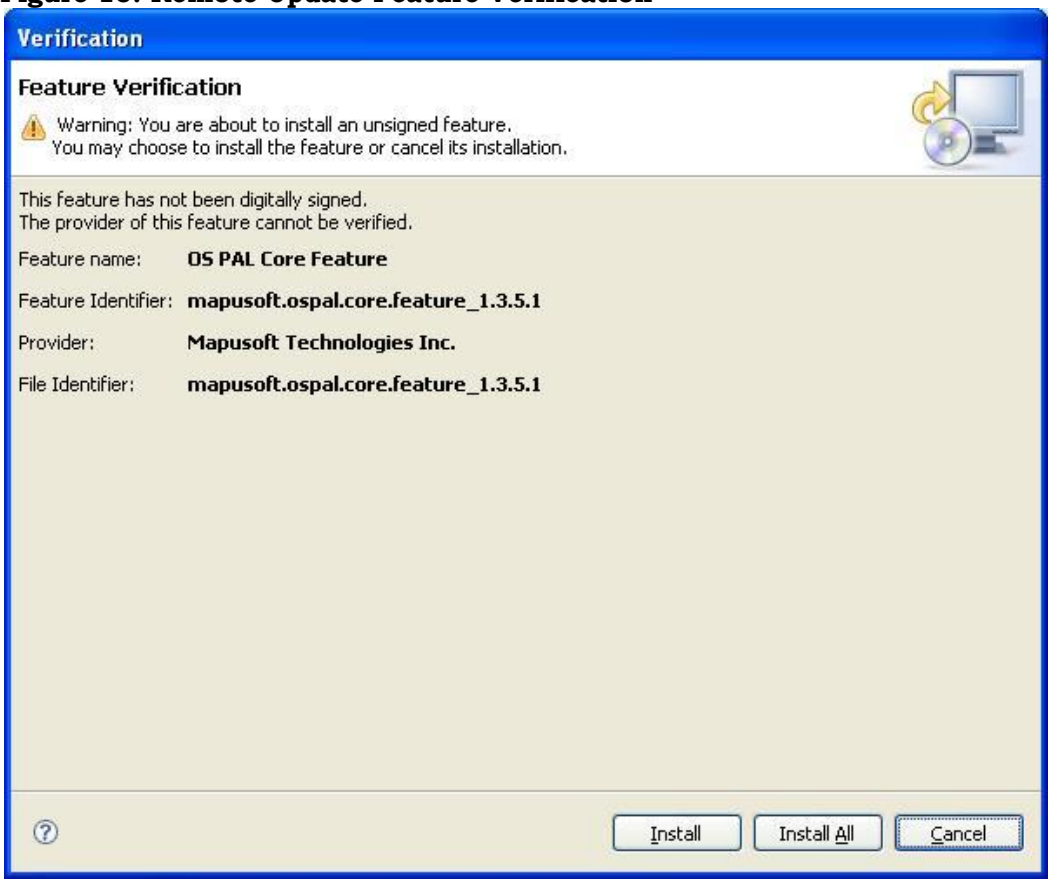

7. On Update Manager window, you can view the new plugins being downloaded as shown in Figure 9.

**Figure 9: Remote Updates Download**



8. On Feature Verification window, click **Install All** as shown in Figure 10.

**Figure 10: Remote Update Feature Verification**

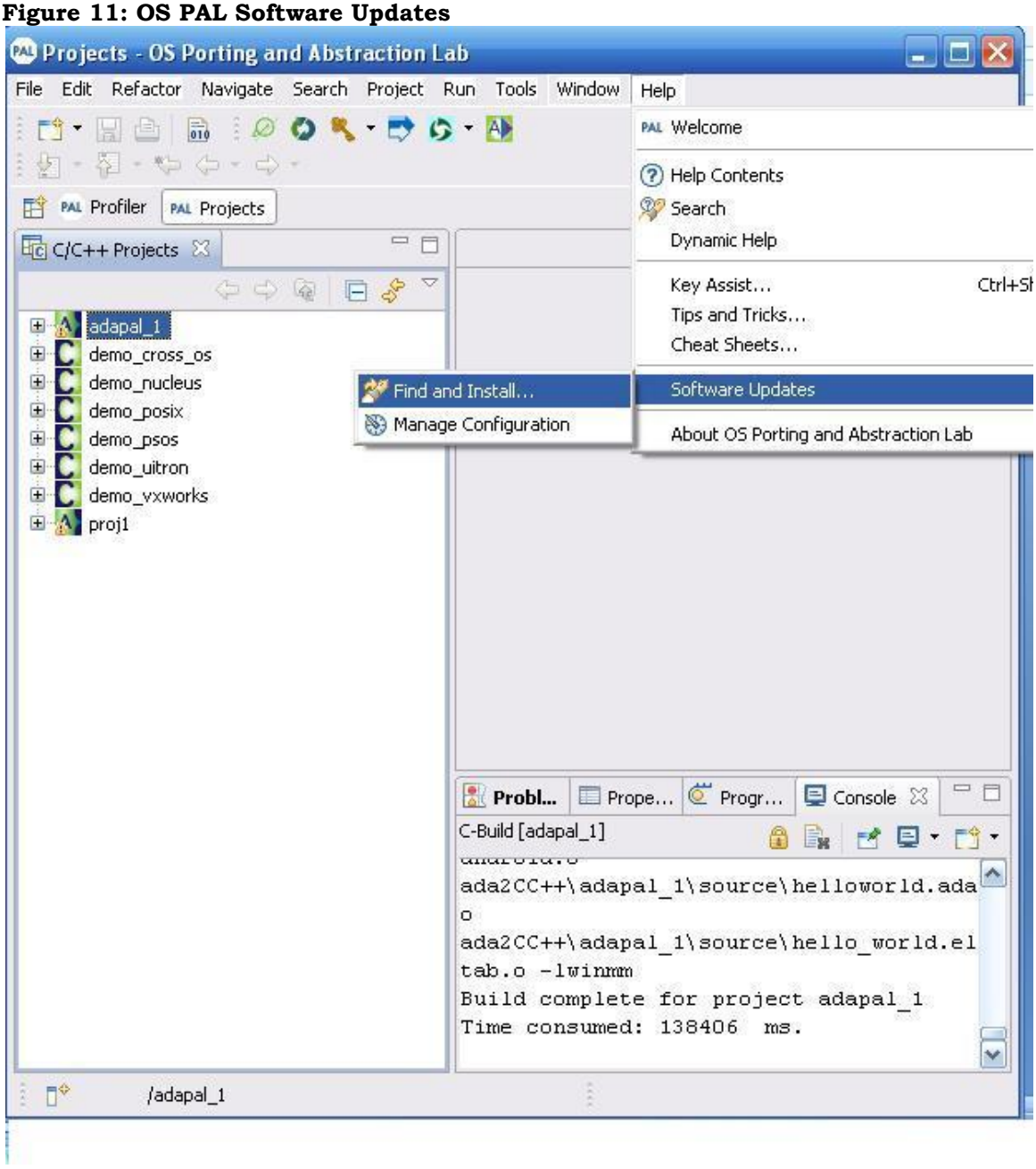



9. Once all the features and plug-ins have been downloaded successfully and their files installed into the product on the local computer, a new configuration that incorporates these features and plug-ins will be formulated. Click **Yes** when asked to exit and restart the Workbench for the changes to take effect. You have now successfully installed new feature updates to your OS PAL using the Remote Update Site.

## Updating Software Using Local Update Site

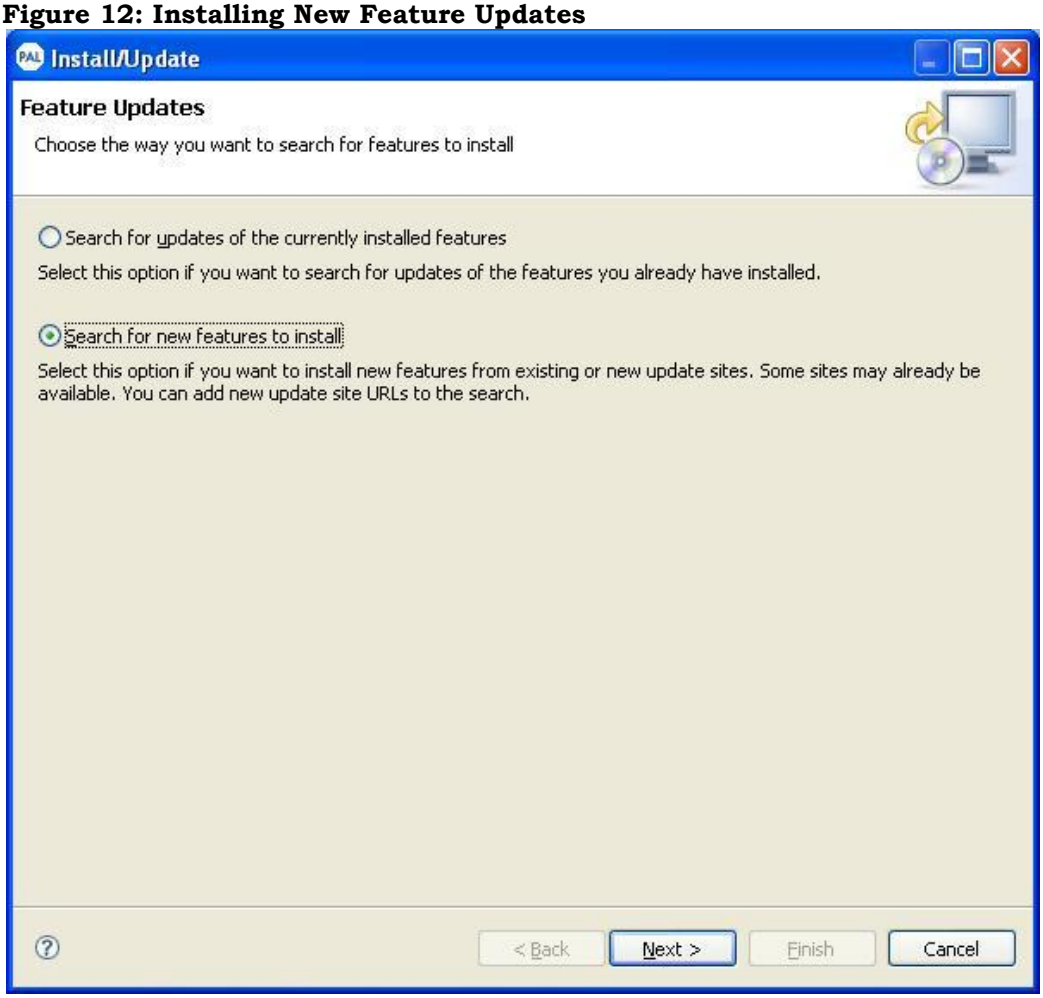
1. From OS PAL main window, select **Help > Software Updates > Find and Install** as shown in Figure 11.

**Figure 11: OS PAL Software Updates**



2. Select **Search for new features to install** and click **Next** as shown in Figure 12.

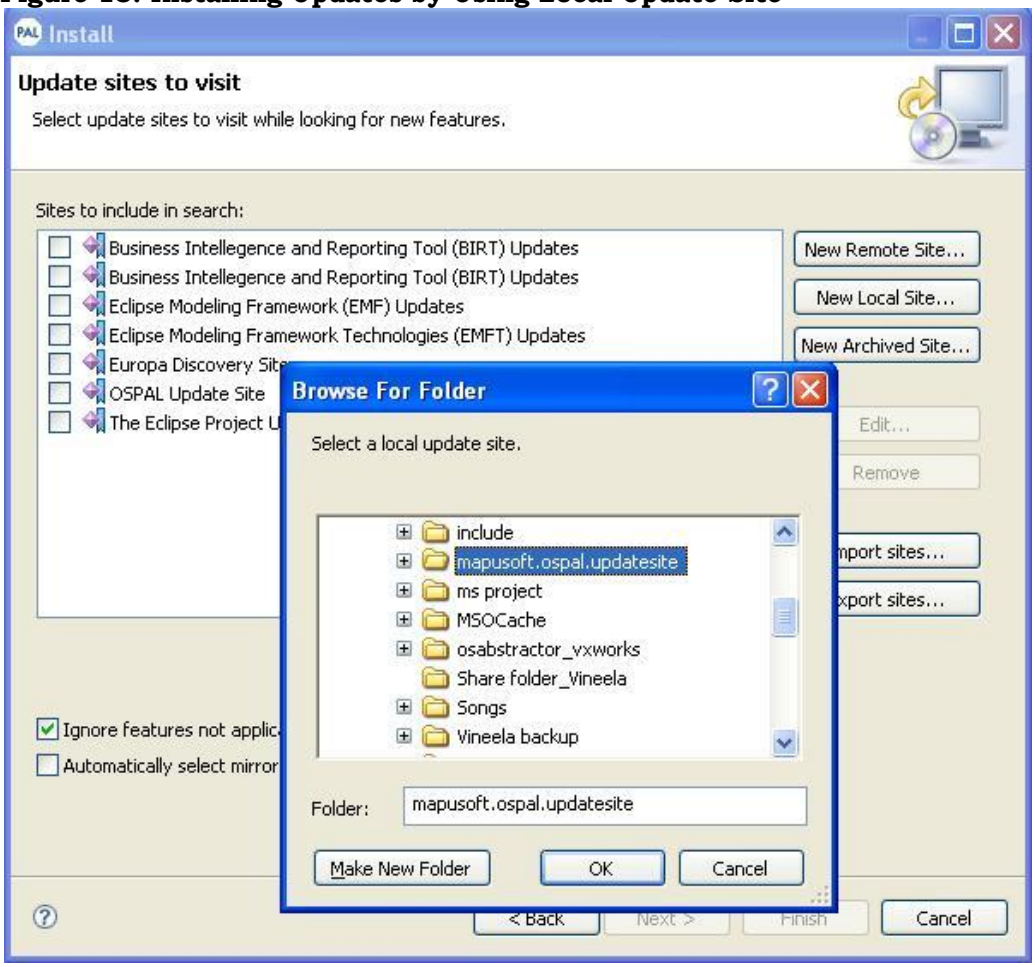
**Figure 12: Installing New Feature Updates**



3. On Updates sites to visit window, select **New Local Site** and browse for the folder provided by MapuSoft, named as *mapusoft. Ospal. update site* and click **OK** as shown in Figure 13.

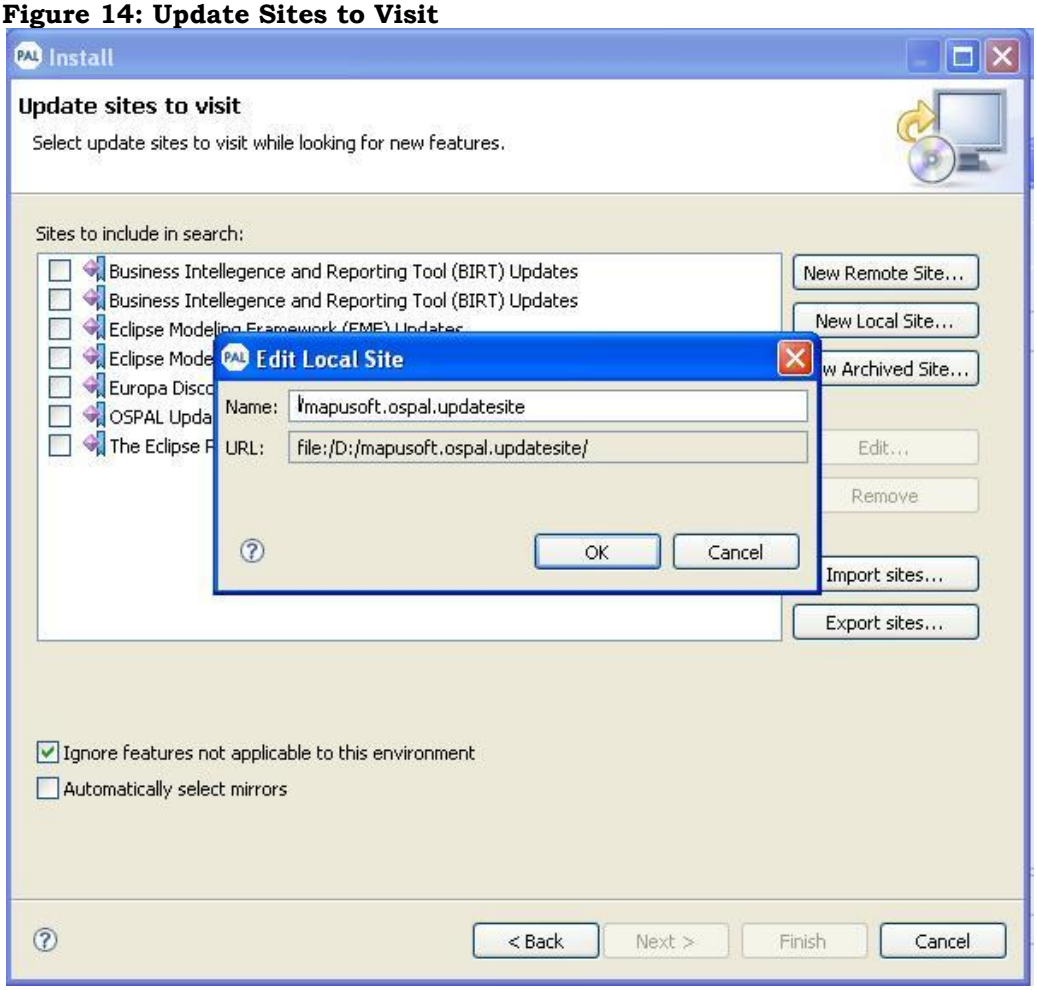
   **NOTE**: If the system does not allow you to give the same site name, select the previous *update site* folder from the list and click **Remove**. Or, you can also save the Update site folder in any other location on your local disk.

**Figure 13: Installing Updates by Using Local Update Site**



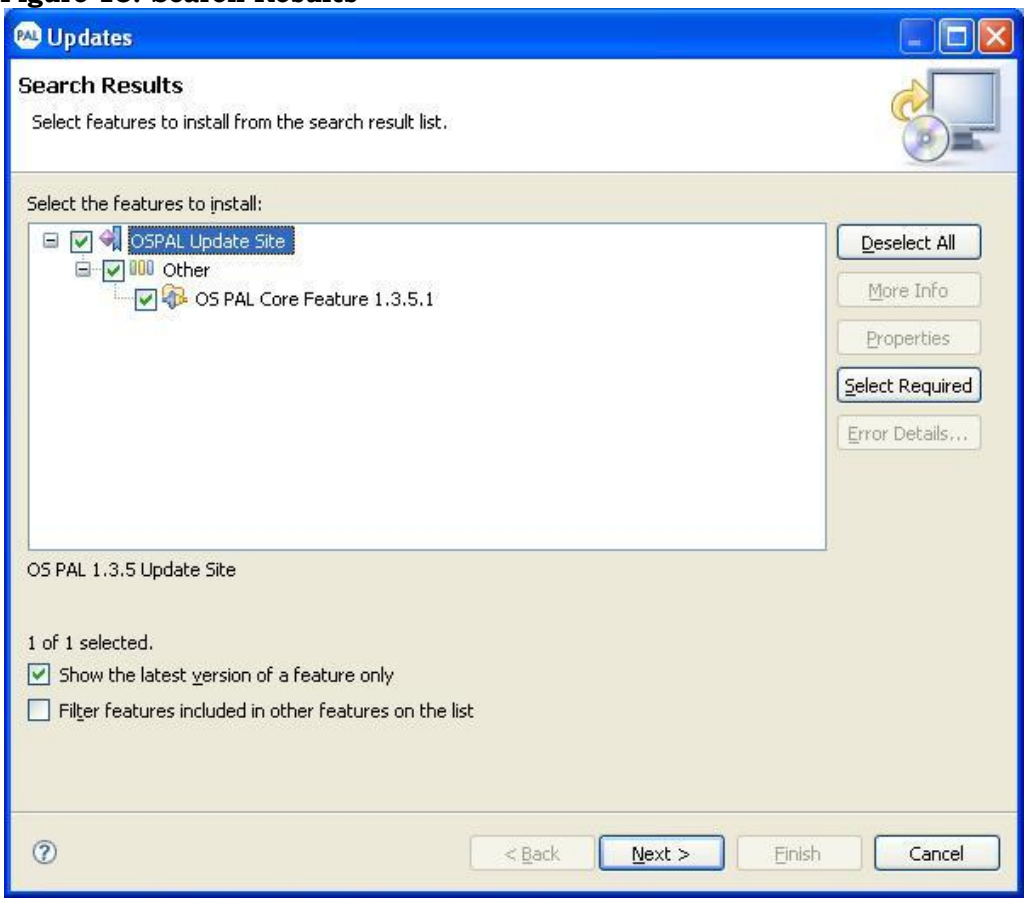
4. On Edit Local Site pop up window, next to **Name** text box, provide a different name and click **OK**. The name can be any name that is not already present on the list as shown in Figure 14 and click **Finish.**

**Figure 14: Update Sites to Visit**



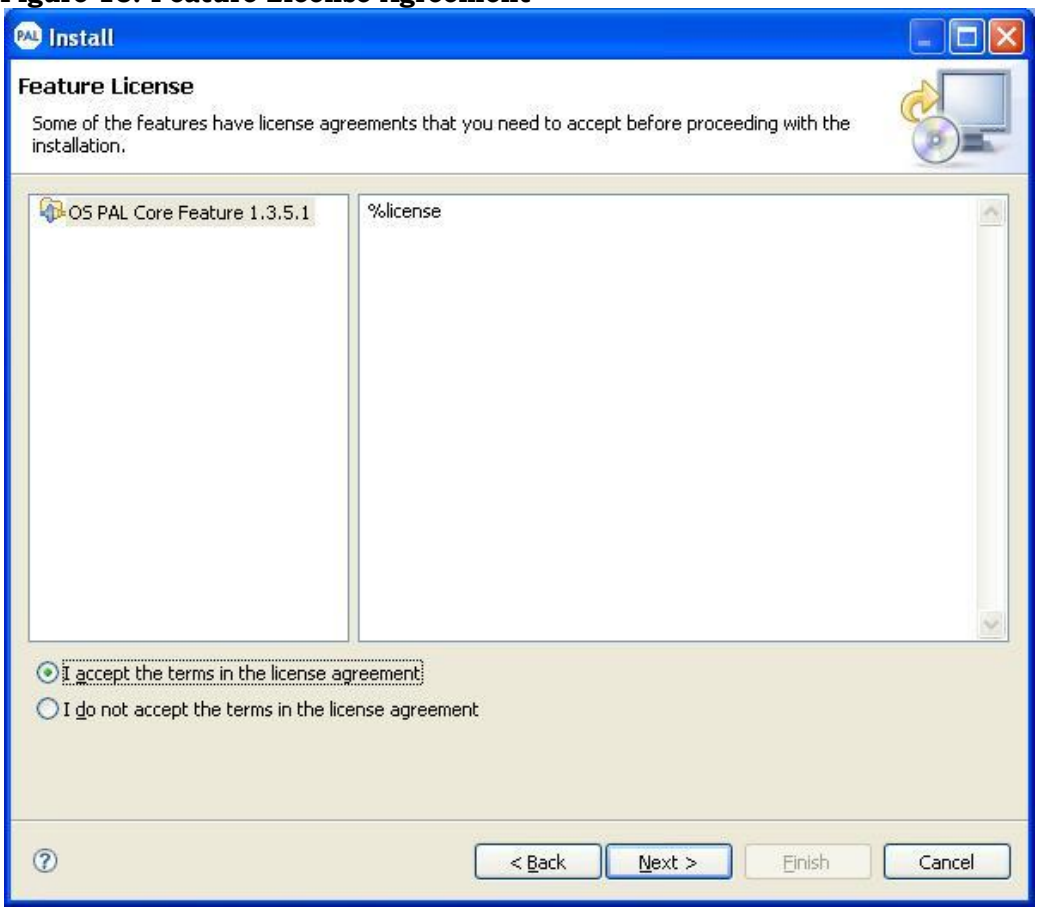
5. On Updates Search Results window, select the features under **Others** tree parent and click **Next** as shown in Figure 15.

**Figure 15: Search Results**



6. On Feature License window, select the radio button next to **I accept the terms in the license agreements** and click **Next** as shown in Figure 16.

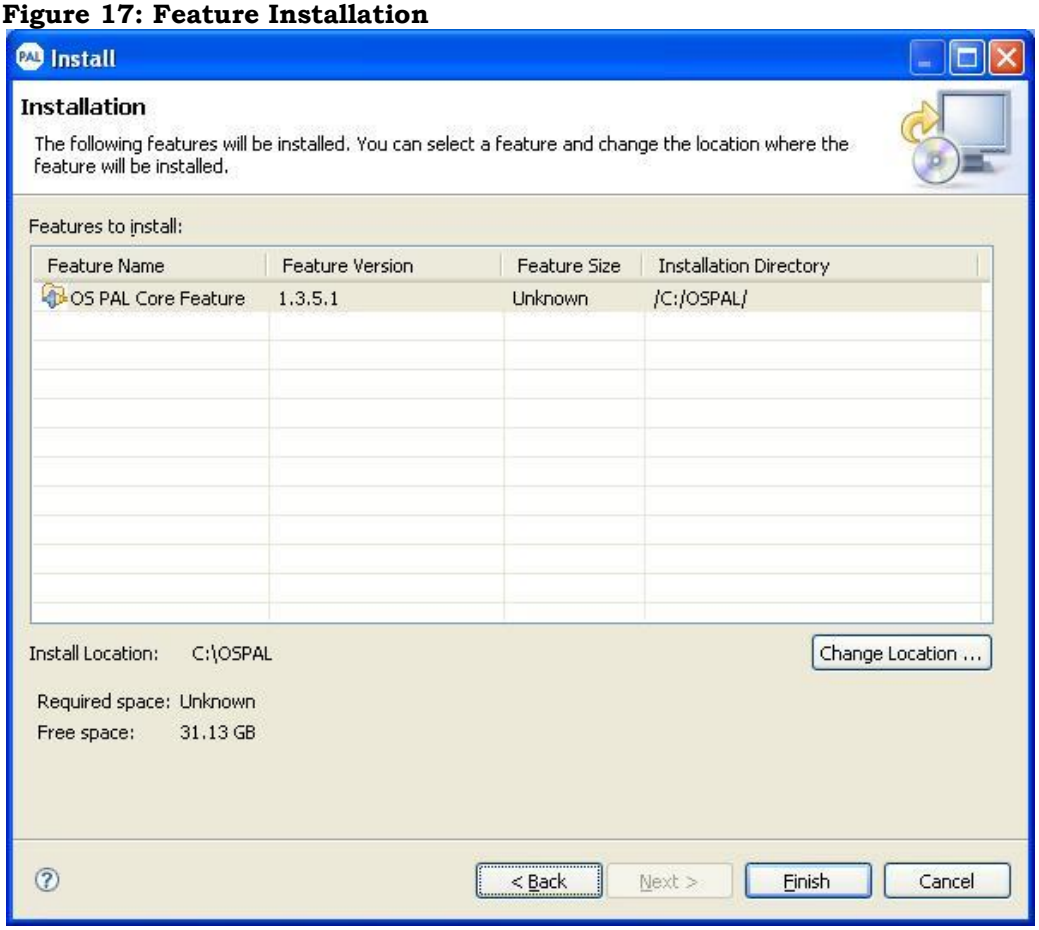
**Figure 16: Feature License Agreement**



7. On Installation window, you can view the features that are going to be installed and the Installation Directory as shown in Figure 17 and click **Finish**.

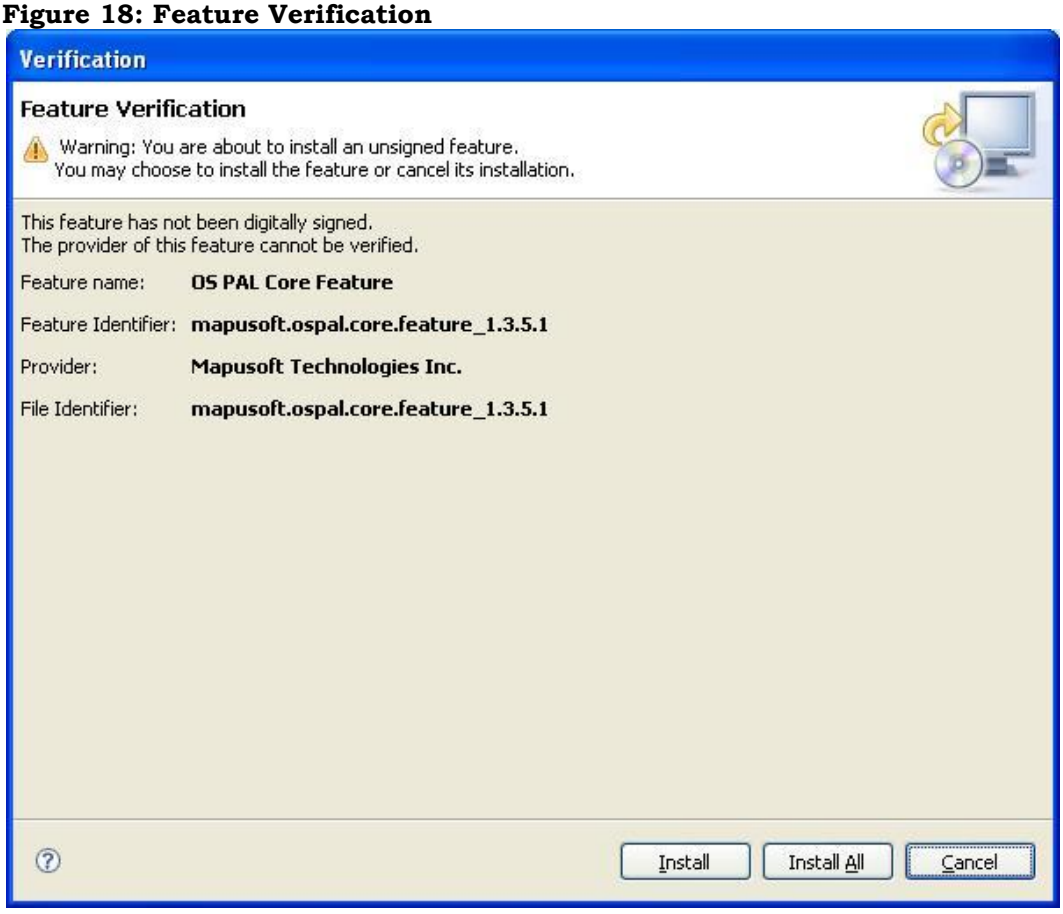
   **NOTE**: You can change the Installation Directory if you want to, by clicking Change location.

**Figure 17: Feature Installation**



8. On Feature Verification window, click **Install All** as shown in Figure 18.

**Figure 18: Feature Verification**




9. Once the new feature and plug-ins have been downloaded successfully and their files installed into the product on the local computer, a new configuration that incorporates these features and plug-ins will be formulated. Click **Yes** when asked to exit and restart the workbench for the changes to take effect.

You have now successfully installed new feature updates to your OS PAL.

# Chapter 2. OS PAL Components

This chapter introduces all the OS PAL components. They are as follows:

Introduction to OS PAL Components
OS Simulator with Host Development

OS Changer for Re-using Software
OS Abstractor for Developing Portable Software
Full Library Package Generator
Optimized Target Code Generator
Ada-C/C++ Changer
About OS PAL Profiler

# Introduction to OS PAL Components

With OS Porting and Abstraction Lab (OS PAL) you can easily port, abstract and optimize your code on a host machine and run the application on different target platforms. OS PAL leverages the existing OS Changer and OS Abstractor technologies while adding advanced code optimization capacities on multiple OS environments. OS PAL provides users an easy-to-use graphical user interface that is integrated with the Eclipse® based CDT environment.

OS PAL uses OS Abstractor and OS Changer technology to produce optimized target code. OS PAL target features include:

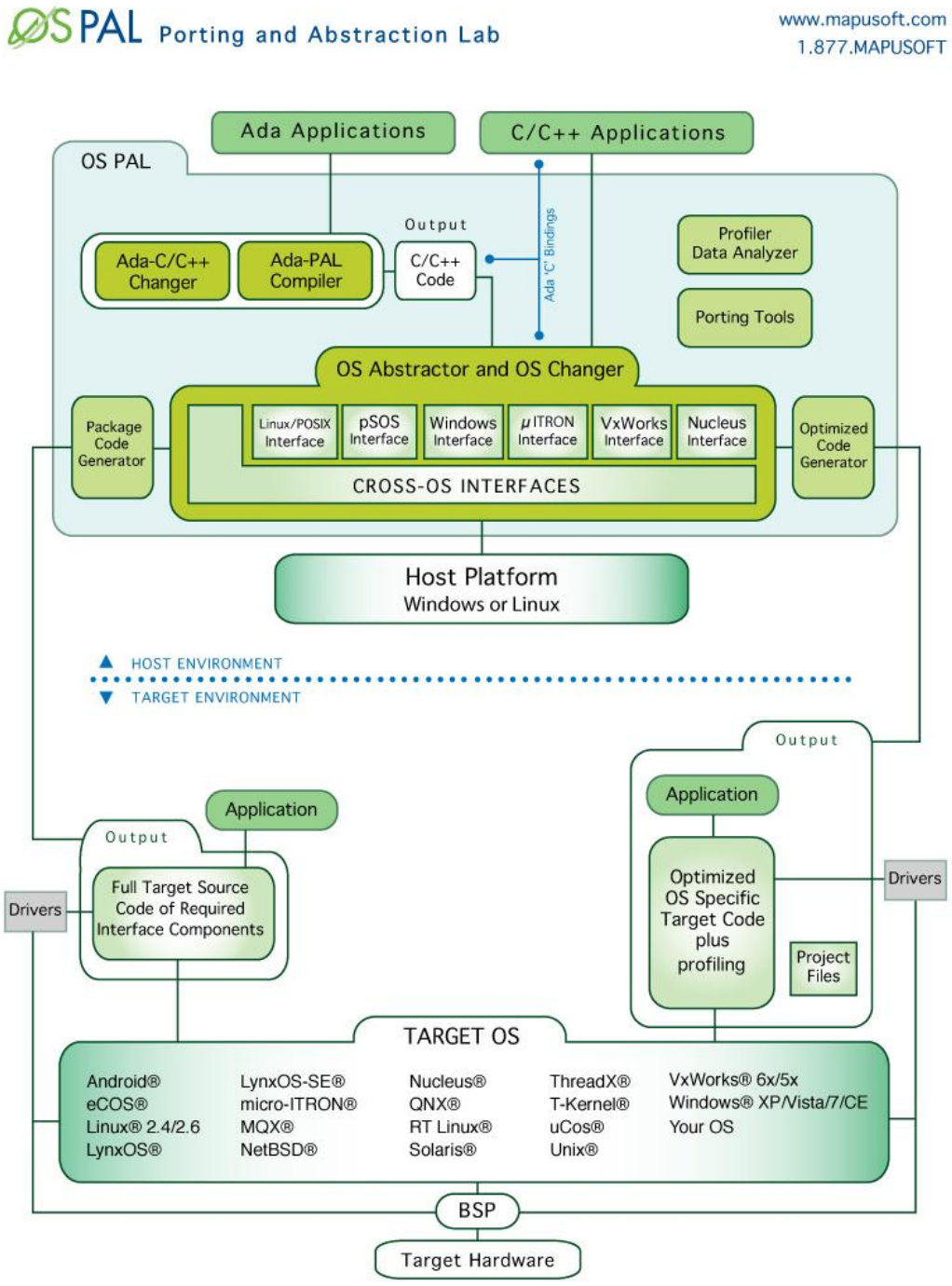
- Generation of project files for your IDE
- Generated target code is optimized to contain only the APIs used by the application
- Allows for further optimization by in-lining user selected API's
- Compile Ada source code to a relocatable object or an executable.
- Enables to convert Ada source code into C/C++ code
- Allows application profiling and platform profiling for your APIs
- Generate API Profiling timing report and Profiling Timing comparison report
- Target selection and configuration tabs to optimize the target code specific for your application
    - Target OS selection
    - Profiler configuration
    - Task configuration including a task pooling feature
    - Process configuration including a process feature
    - Memory configuration
    - Resource configuration
    - Debug configuration
    - Output configuration including the ability to output to a console or serial port
    - ANSI Mapping configuration
    - Device I/O configuration

MapuSoft provides an illustration to describe all the components of OS PAL. OS PAL leverages the existing OS Changer and OS Abstractor technologies while adding advanced code optimization capacities on multiple OS environments. They are all interlinked and work closely as shown in Figure 19.

**Figure 19: OS PAL Components**


For more information on OS PAL components, refer to the specific chapter.

# OS PAL Architecture



# OS Simulator with Host Development

OS PAL simulates various OS interfaces such as VxWorks, pSOS, POSIX and Nucleus on host development environments so users can develop embedded code with preferred OS APIs and without the target hardware. OS PAL's state-of-the-art Eclipse based IDE offers seamless integration into existing development flows.

With OS Porting and Abstraction Lab (OS PAL) you can easily port, abstract and optimize your code on a host machine and run the application on different target platforms. OS PAL leverages the existing OS Changer and OS Abstractor technologies while adding advanced code optimization capacities on multiple OS environments. OS PAL provides users an easy-to-use graphical user interface that is integrated with the Eclipse® based CDT environment. Target operating systems supported can be found here: http://mapusoft.com/products.

OS Simulator with Host Development chapter includes the following topics:

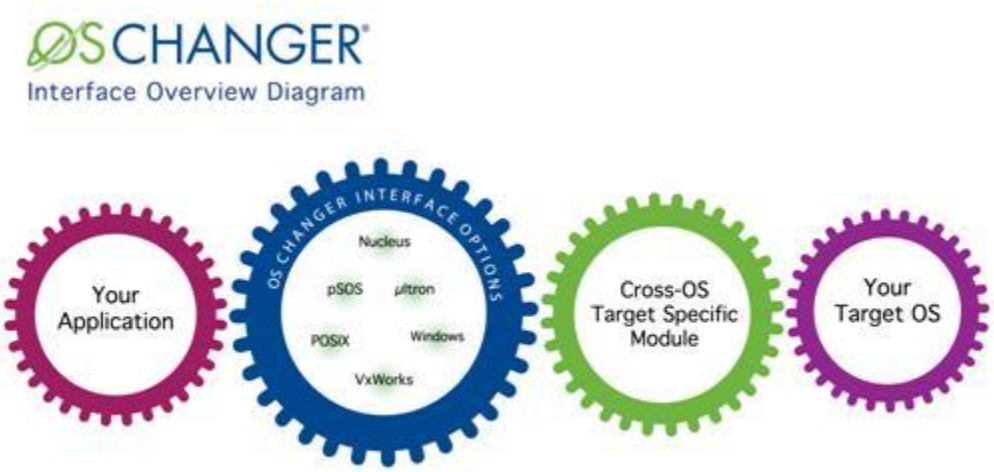
- List of Available OS Simulators
- Host Development Environment
- Creating an OS PAL C/C++ Project
- C Project Template Files
- HOST Defines
- Adding Source Code Files to OS PAL Project
- Building Binary Files for a Project
- Executing Binary Files
- Debugging the Demos Supplied by MapuSoft
- Debugging Using External Console/Terminal
- Inserting Application Code to Run only on Host Environment
- Inserting Application Code to Run only on Specific Target OS Environment
- Updating Project Settings

For more information on the host development refer to OS Simulator With Host Development Chapter on page 37.

# OS Changer for Re-using Software

The OS Changer family of products is COTS porting tools that give users the freedom to change operating systems while reusing their existing embedded code and knowledge base to protect their software investment and avoid costly porting issues. OS Changer also allows developers to write code using a familiar application programming interface (API) and run the application on a wide variety of supported target OS platforms. Solutions are available for porting from VxWorks, pSOS, and Nucleus to many different real time (RTOS) and non-real time operating systems. Target operating systems supported can be found here: http://mapusoft.com/products/.

OS Changer is designed for use as a C library. Services used inside your application software are extracted from the OS Abstractor libraries and are combined with the other application objects to produce the complete image. OS Changer is graphically represented in the following diagram.




OS Changer for reusing software chapter includes the following topics:

- About OS Changer
- Interfaces Available for OS Changer
- How to Use OS Changer
- Conditional Compilations
- OS Changer Defines
- Error Handling

For more information on OS Changer refer to OS Changer for Reusing the Code chapter on page 112.

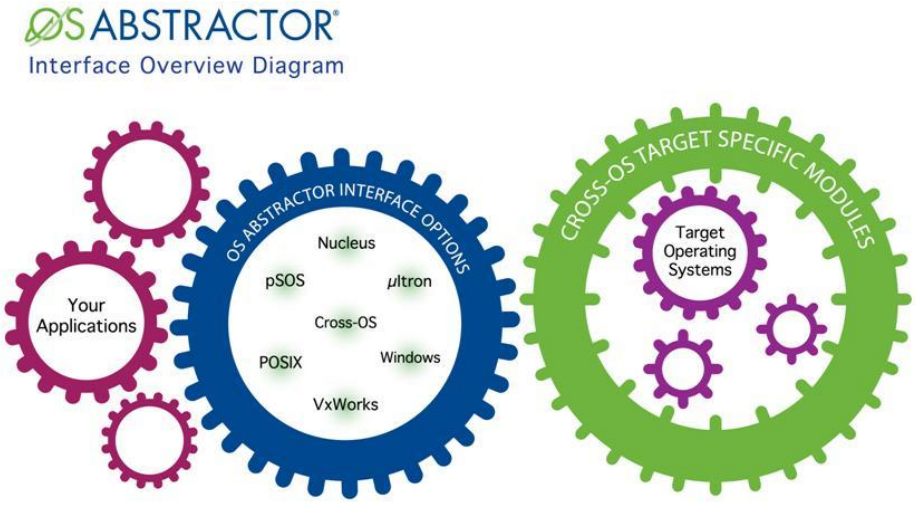
# OS Abstractor for Developing Portable Software

Developing a solid software architecture that can run on multiple operating systems requires considerable planning, development and testing as well as upfront costs associated with the purchase of various OS and tools to validate your software. MapuSoft's OS Abstractor is an effective and economical software abstraction alternative for your embedded programming. By using OS Abstractor, your embedded application can run on many real time (RTOS) and non-real time operating systems to negate any porting issues in the future when your platform changes. Target operating systems supported can be found here: http://mapusoft.com/products.

Cross-OS interface provides you a robust and standard OS interface architecture for flexible application development and portability while eliminating the risks associated with selecting an OS and dependency on a single vendor. OS Abstractor makes your application adapt to multiple operating system platforms with a standard OS interface, thereby reducing cost associated with code maintenance and learning multiple operating systems.

OS Abstractor is designed for use as a fully scalable C library. Services used inside your application software are extracted from the OS Abstractor libraries and are combined with the other application objects to produce the complete image. This image may be downloaded to the target platform or placed in ROM on the target platform. Application developers need to specify the OS for the application and also include the required OS Abstractor libraries while building the application. Application developers can also select the individual OS Abstractor components that are needed and exclude the ones that are not required.

OS Abstractor is graphically represented in the following diagram.




OS Abstractor for Developing Portable Software includes the following topics**:**

- Introduction to OS Abstractor Products
- Installing OS Abstractor Products
- Using OS Abstractor

For more information on this, refer to the OS Abstractor For Developing Portable Software Chapter on page 116.

# Full Library Package Generator

With OS Porting and Abstraction Lab (OS PAL) you can easily generate a source code package to create libraries and develop application using your own IDE.

You can manually scale and configure the product by modifying the user configuration file.

OS PAL provides users an easy-to-use graphical user interface that is integrated with the Eclipse® based CDT environment. Target operating systems supported can be found here: http://mapusoft.com/products/.

Full Source Library Package Generator chapter includes the following topics:

- Generating Full Library Packages
- How to Use Libraries With Your Application

For more information on full source library package generator, refer to Full Library Package Generator chapter on page 120.

# Optimized Target Code Generator

With OS Porting and Abstraction Lab (OS PAL) you can easily port, abstract and optimize your code on a host machine and run the application on different target platforms. OS PAL leverages the existing OS Changer and OS Abstractor technologies while adding advanced code optimization capacities on multiple OS environments. OS PAL provides users an easy-to-use graphical user interface that is integrated with the Eclipse® based CDT environment. Target operating systems supported can be found here: http://mapusoft.com/products/.

OS PAL reads application source code to determine the services used by your application and produces OS specific interface code optimized for your specific application and for each target OS platform. OS PAL gives you the ability to support multiple OS. It is also easily expandable to generate code for your proprietary OS.

Optimized Target Code Generator chapter includes the following topics:

- Generating Target Code
- Generating Project Files for your Target
- Running OS PAL Generated Code on your Target

For more information on optimized target source code generator, refer to Optimized Target Code Generator chapter on page 128.

# Ada-PAL Compiler

The Ada-PAL compiler translates Ada source programs into re-locatable object modules and records dependency information for use by the program builder. It optionally generates source listing, assembly listing and debugger information for use by the symbolic debugger. The Ada-PAL compiler consists of two phases—the front end and the back end. The front end performs syntactic and semantic analysis. It generates C source files as input to the back end. The back end of the Ada-PAL compiler is an ISO/ANSIC compiler. It performs code generation, applies optimizations, and generates a re-locatable object module.

The importing directory which consists of Ada sources that needs to be converted to C Sources. This may be referred also as Ada Program Library. The Ada program library contains all information needed to support the separate compilation requirements of Ada. The primary contents of the program library are Ada source files and object modules created by the compiler. The only additional information maintained by the program library is the correlation between unit names and source files, and dependency information associated with object modules.

For more information on Ada-PAL compiler, refer to Using Ada-PAL Compiler chapter.

# Ada-C/C++ Changer

MapuSoftTechnologies now offers the Ada to C conversion Ada tool to give developers the ability to automatically convert legacy software written in Ada to the C programming language. This automatic code conversion process eliminates the need for a costly and tedious code re-write to provide developers extensive cost and time savings. Ada tool gives users peace of mind by providing an error free tool that prevents mistakes made in the error prone task of a manual rewrite. Ada tool supports converting Ada 83 and Ada 95 source code and generates ANSI C output as well as certain C++ features while preserving the Ada code's comments, files structures and variable names to ease ongoing code maintenance.

For more information on using Ada to C/C++ Changer, refer to Ada to C/C++ Changer chapter on page 210.

# Profiling Applications and Platform

OS PAL enables you to view API performance data

- The OS PAL Profiler feature enables API data collection
- Collected data provides feedback concerning the utilization of MapuSoft's APIs in the project
- Reports allow for performance impact analysis by detailing API execution time
- Offers area, bar, line, pie and scatter charts for data analysis
- Generate API timing report and Timing comparison report
- Platform API Profiling–System specific API profiling
- Application Profiling–User specific API profiling

**Platforms Supported for OS PAL Profiler**

- VxWorks 6x® and VxWorks 5x®
- Linux 2.4® and Linux 2.6®
- LynxOS® and LynxOS-SE®
- Solaris
- Unix®
- Windows CE®
- Windows XP®
- QNX®

For more information on Profiling, refer to OS PAL Profiler chapter on page 167.

# Chapter 3. OS Simulator with Host Development

This chapter contains the following topics:

List of Available OS Simulators

Host Development Environment

Creating an OS PAL C/C++ Project

Adding Source Code Files to OS PAL Project

Building Binary Files for a Project

Executing Binary Files

Debugging the Demos Supplied by MapuSoft

Debugging Using External Console/Terminal

Inserting Application Code to Run only on Host Environment

# List of Available OS Simulators

The following is the list of available OS Simulators:

- VxWorks
- pSOS
- Nucleus
- POSIX
- uITRON
- Windows

# Host Development Environment

Host development needs a proper environment to run and build embedded programs. To develop an environment you need the following GNU tools:

- Eclipse IDE
- MinGW
- GNU Compiler
- PAL Debugger

## Eclipse

An IDE is a powerful set of tools in the OS Porting and Abstraction Lab (PAL) development suite. The IDE is based on the Eclipse Platform developed by Eclipse.org, an open consortium of tools vendors.

The IDE incorporates into the Eclipse framework several OS PAL-specific plugins designed for building projects for target systems running on HOST. The tools suite provides a single, consistent, integrated environment, regardless of the host platform you are using Windows, Linux, Solaris*. Plugins from most vendors should work within the Eclipse framework in the same way.

**NOTE**: For more information on Eclipse and working on Eclipse framework, refer to http://www.eclipse.org/documentation/.

## MinGW

MinGW, a contraction of "Minimalist GNU for Windows", is a port of the GNU Compiler Collection (GCC), and GNU Binutils, for use in the development of native Microsoft Windows applications. Offered in easily installed binary package format, for native deployment on MS-Windows, or user-built from source, for cross-hosted use on UNIX or GNU/Linux, the suite exploits Microsoft's standard system DLLs to provide the C-Runtime and Windows API. It is augmented by additional function libraries for improved ISO C-99 compatibility, and further, by community supported add-on tools and libraries, many pre-built, many more in the form of "mingw PORTs", to be built by the end user.

MinGW provides a complete Open Source programming tool set which is suitable for the development of native MS-Windows applications, and which do not depend on any 3rd-party C-Runtime DLLs.

## GNU Compiler

The GNU Compiler Collection includes front ends for C, C++, Java as well as libraries for these languages (such as libstdc++, libgcj).

## OS PAL Supplied GDB

A debugger is a computer program that is used to test and debug other programs (the "target" program). The code to be examined might alternatively be running on an instruction set simulator (ISS), a technique that allows great power in its ability to halt when specific conditions are encountered but which will typically be somewhat slower than executing the code directly on the appropriate processor. Some debuggers offer two modes of operation - full or partial simulation to limit this impact.

Typically, debuggers also offer more sophisticated functions such as running a program step by step (single-stepping or program animation), stopping (breaking) (pausing the program to examine the current state) at some event or specified instruction by means of a breakpoint, and tracking the values of some variables. Some debuggers have the ability to modify the state of the program while it is running, rather than merely to observe it. It may also be possible to continue execution at a different location in the program.

The GNU Debugger, usually called just GDB and named gdb as an executable file, is the standard debugger for the GNU software system. It is a portable debugger that runs on many Unix-like systems and works for many programming languages.

While working on OS PAL, you must use MapuSoft's GNU Debugger, called as "OS PAL Supplied GDB".

# Creating an OS PAL C/C++ Project

**NOTE**: This feature requires a license. Click http://mapusoft.com/downloads/ospal-evaluation/ to request an evaluation license.

**NOTE 1**: In New C projects creation, the flag OS_CPU_64BIT will be set to OS_FALSE by default and user needs to make this true if they run on a 64-bit OS.

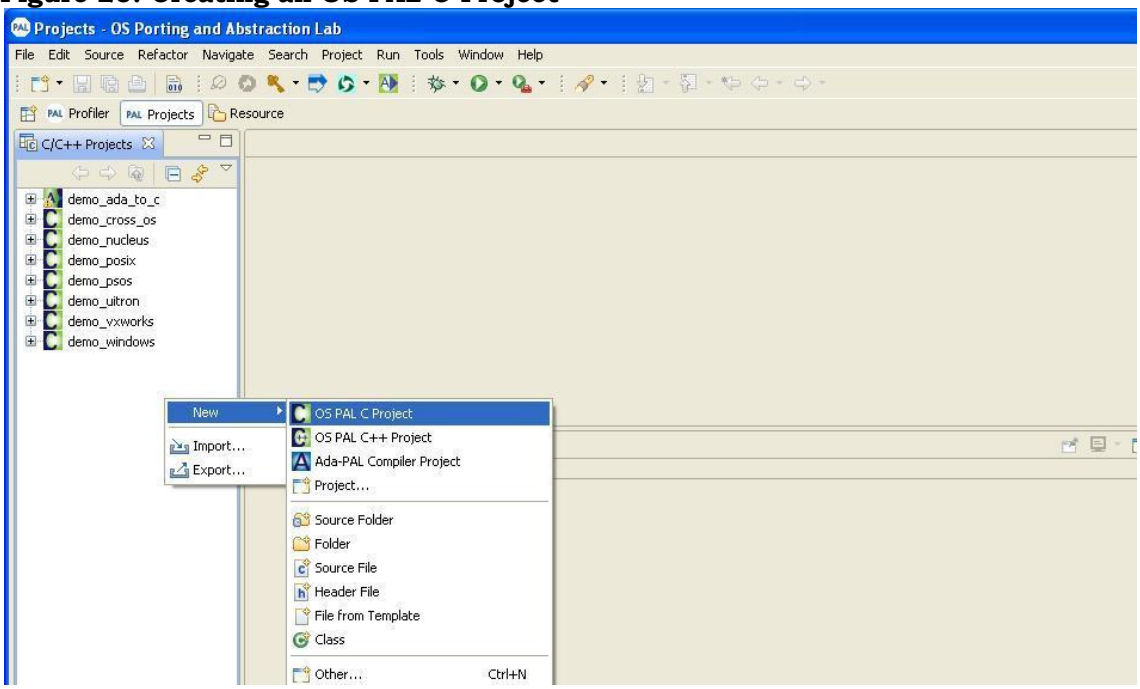
**NOTE 2:** For creating new project, OS_HOST flag is set to OS_TRUE.

**NOTE 3**: When you are working on 64bit architecture, make sure that -m32 flag is added to both the compiler and linker options in project properties to avoid compilation errors

To create an OS PAL C/C++ project:

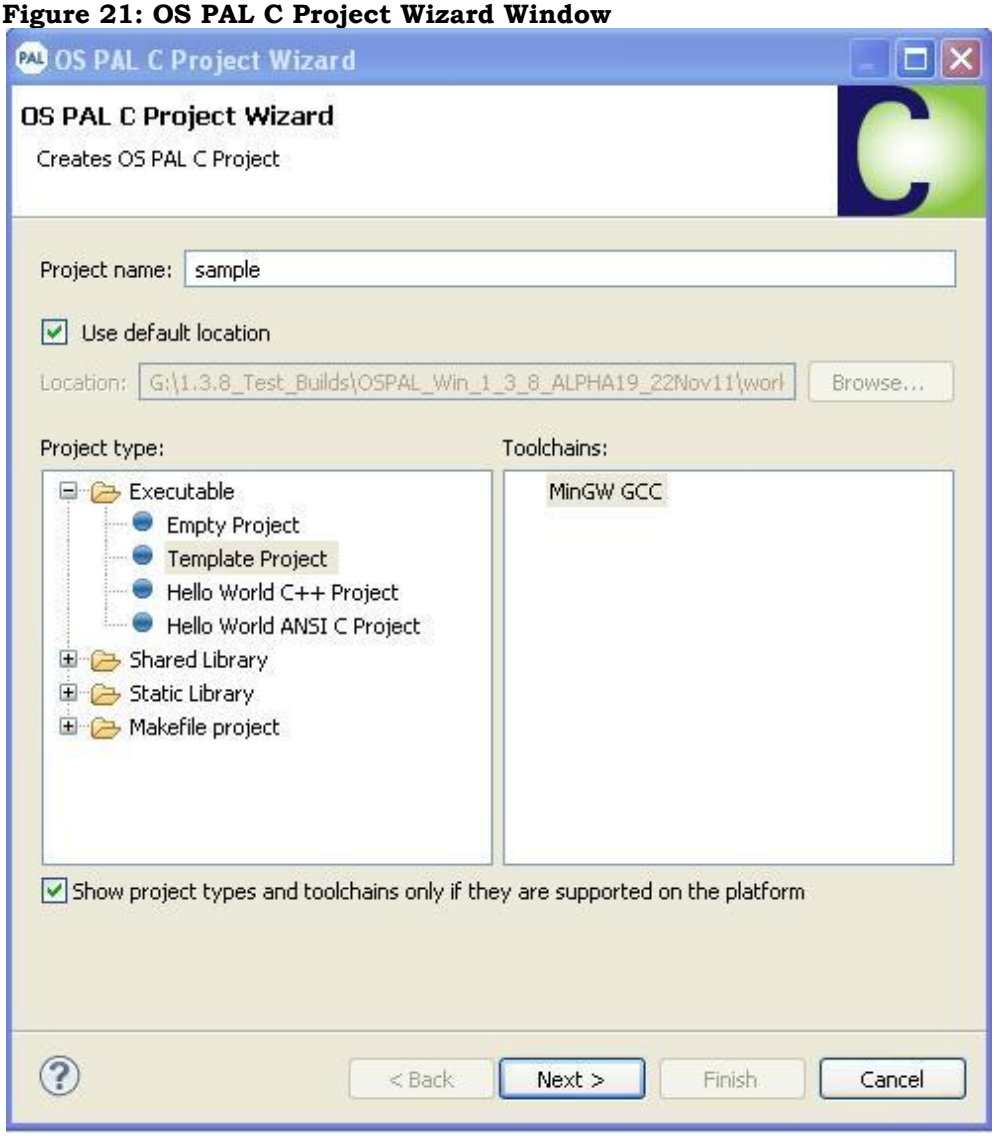
1. From OS PAL main window, select any project under C/C++ **Projects** tab on the left pane.
2. Select **New > OS PAL C/C++ Project** as shown in Figure 20.

**Figure 20: Creating an OS PAL C Project**



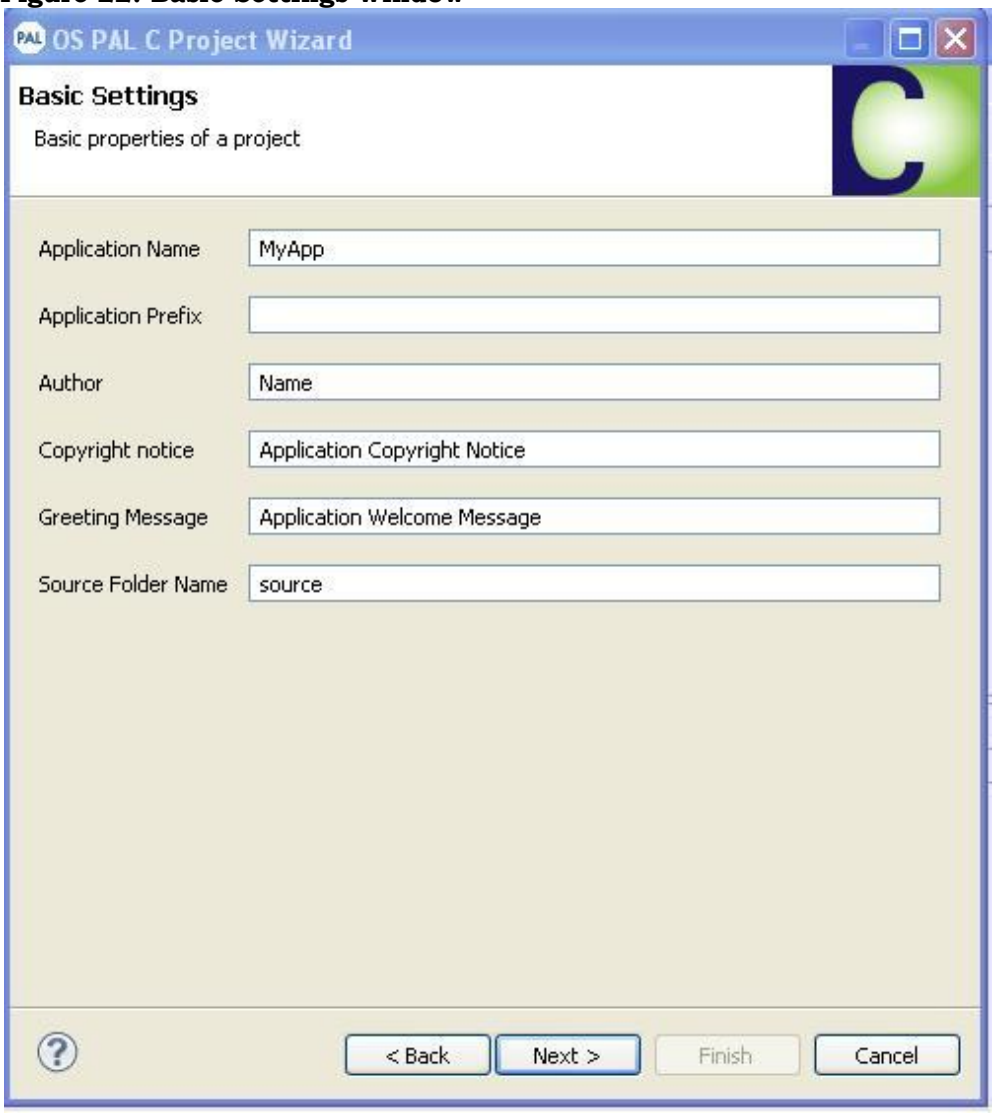
3. On OS PAL C/C++ Project Wizard window, type a project name and give a location next to **Project Name** text box.
4. Under Project Types, expand the **Executable** menu. Select **Template Project** and click **Next** as shown in Figure 21.

**Figure 21: OS PAL C Project Wizard Window**



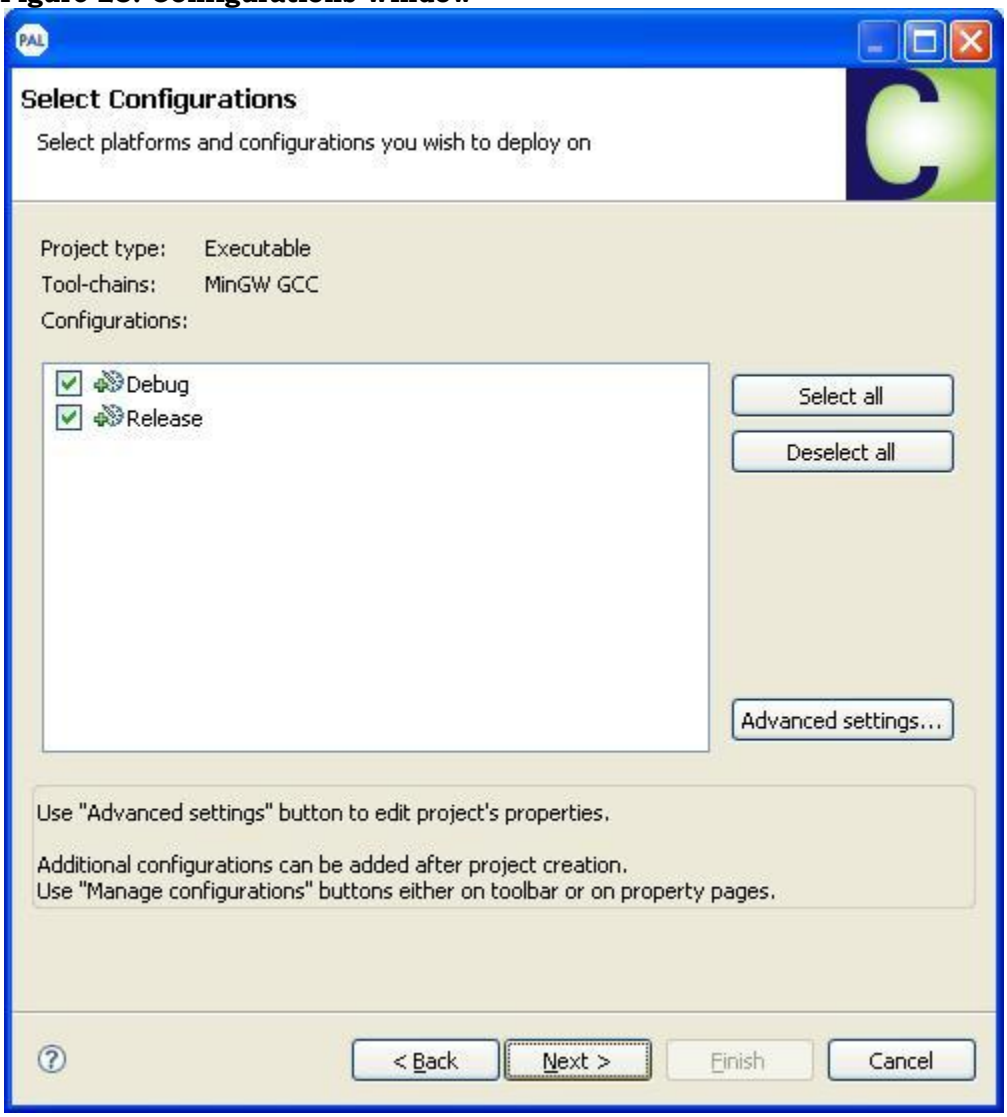
5. On Basic Settings window, define the basic properties of your project and click **Next** as shown in Figure 22.

**Figure 22: Basic Settings Window**



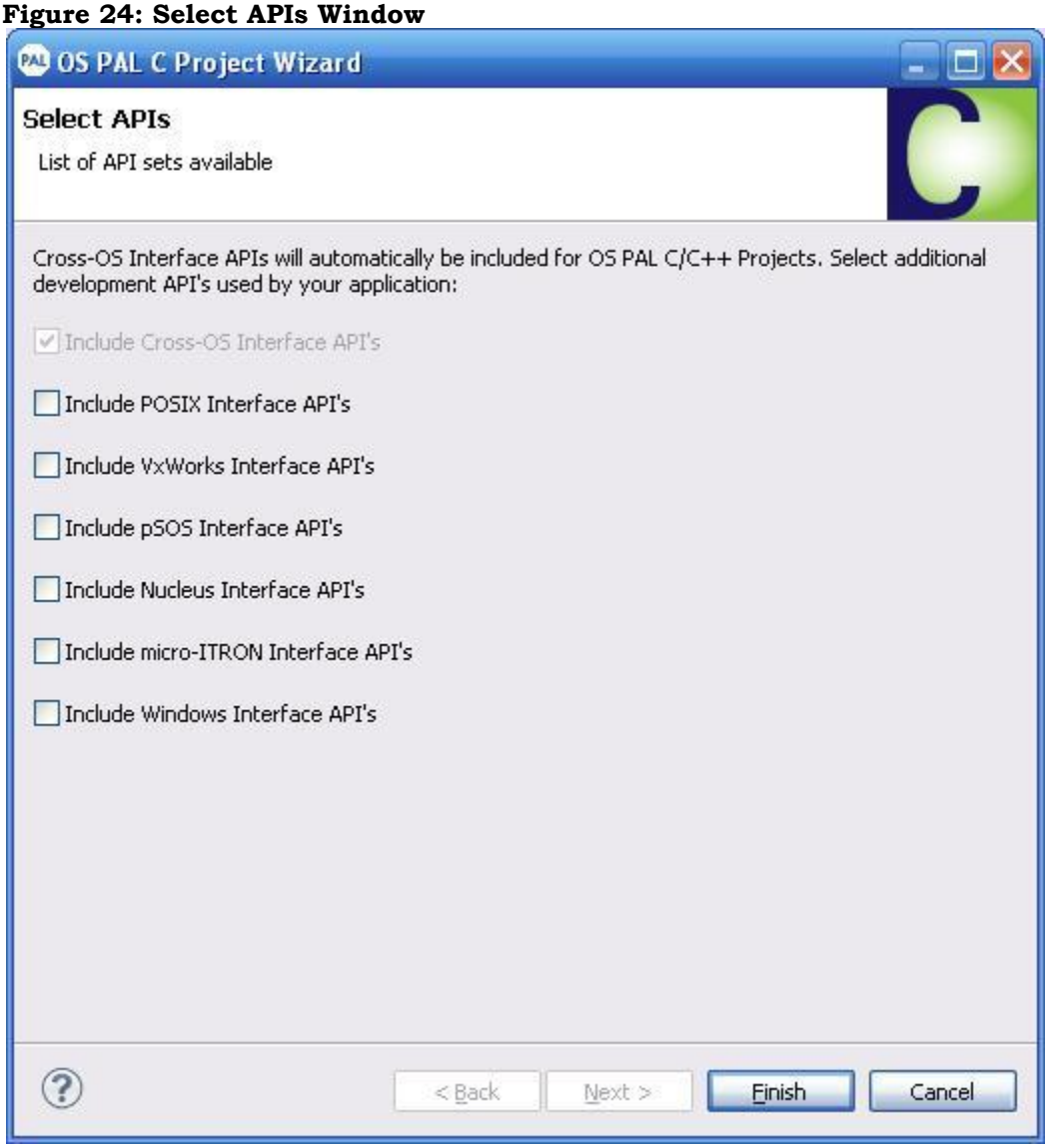
6. On Select Configurations window, select the platforms and configurations for deployment and click **Next** as shown in Figure 23.

**Figure 23: Configurations Window**



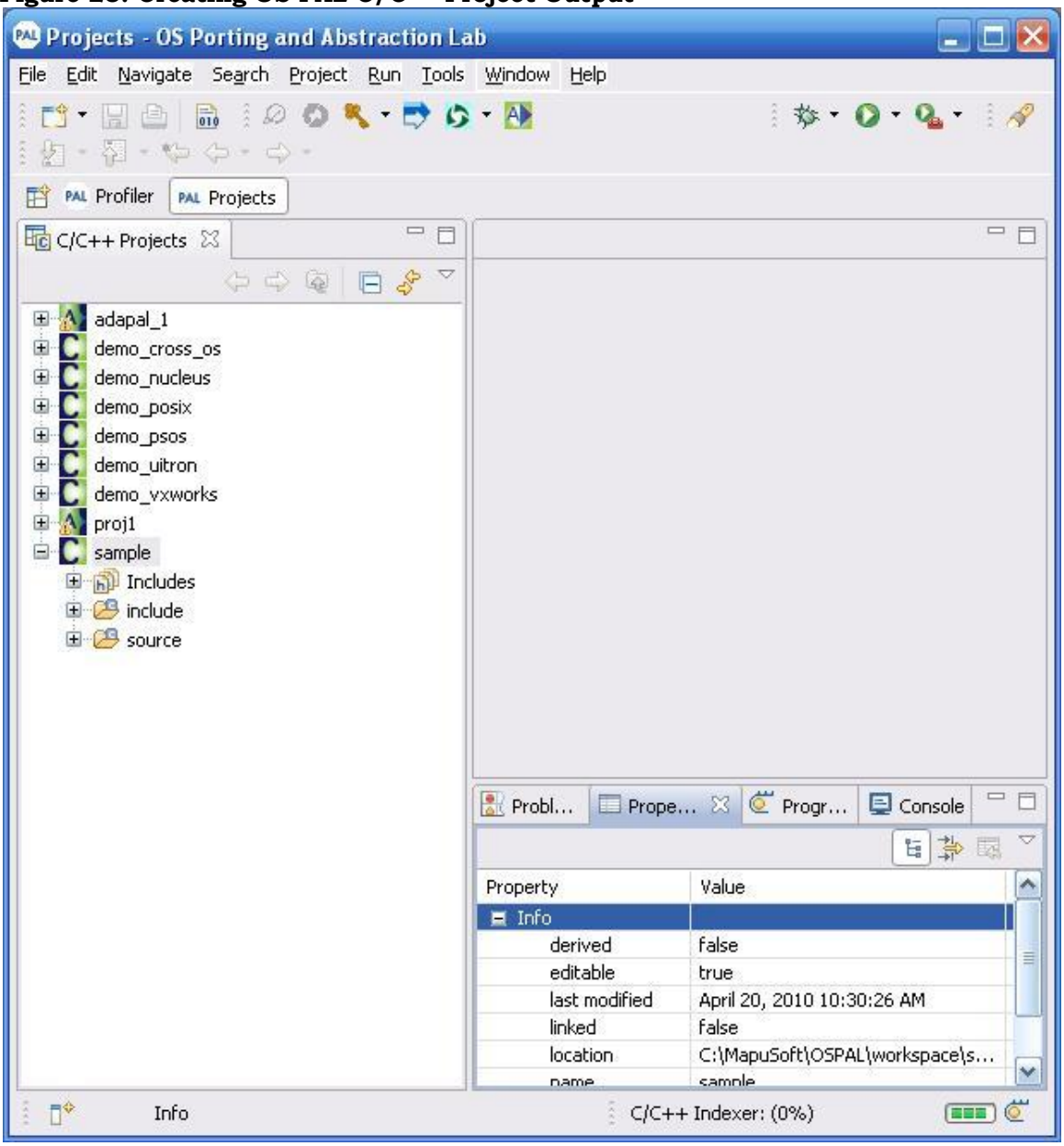
7. On Select APIs window, select the required OS PAL development APIs and click **Finish** as shown in Figure 24.

**Figure 24: Select APIs Window**



You will see the output as shown in Figure 25.

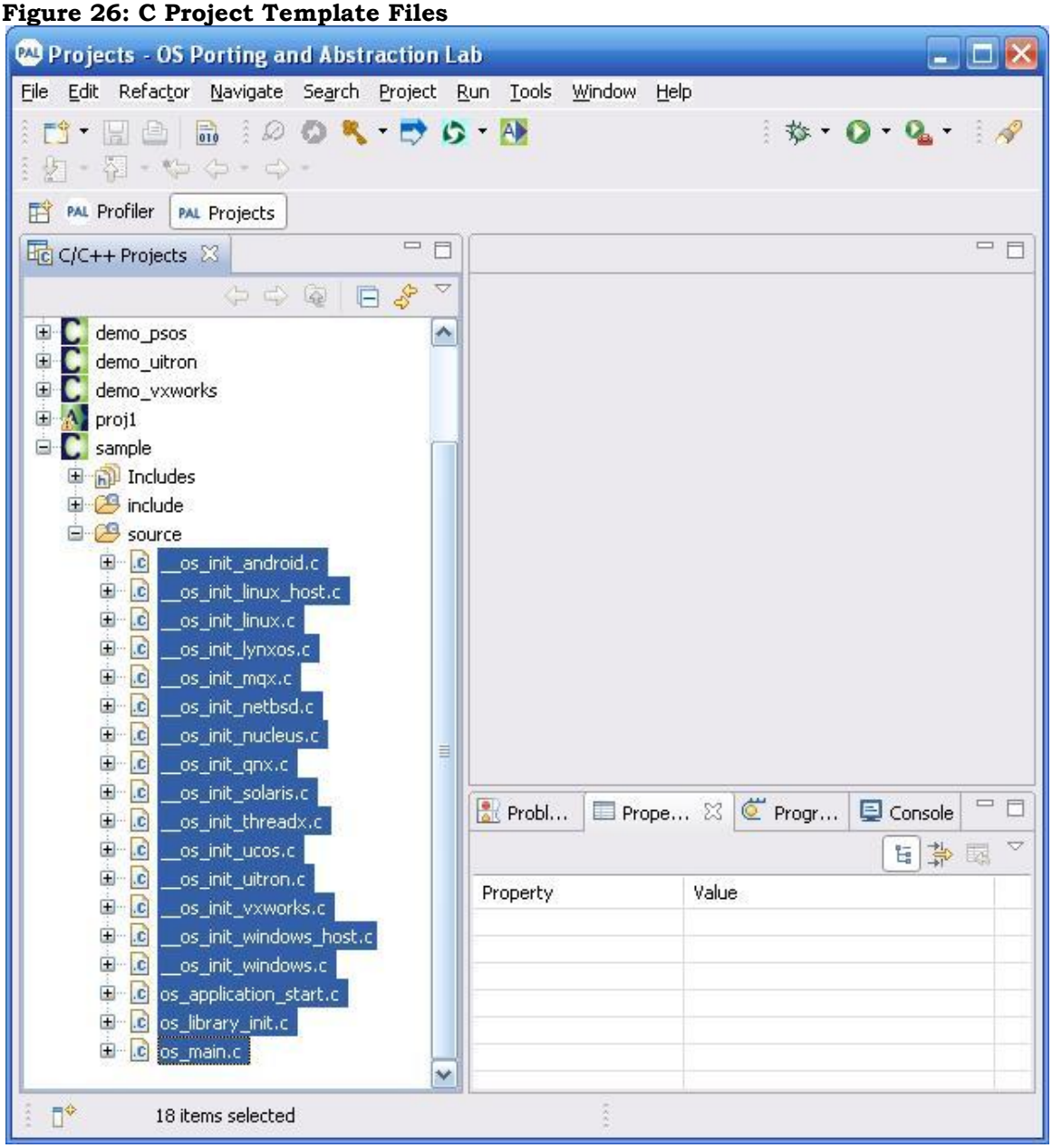
**Figure 25: Creating OS PAL C/C++ Project Output**



## OS PAL Project Template Files

To view the OSPAL C project template files, expand the project folder you have just created by clicking on the +sign beside the Project name as shown in Figure 26**.**

**Figure 26: C Project Template Files**




You can view the following template files for your project on the left pane of the window:

- **__os_init__linux_host.c**–This function is the entry function for the native operating system.This is where you should put any of your operating system specific code. For instance, if you want to add a signal handler on Linux host, you could do it here before calling OS_Main().
- **__os_init_linux.c–**This function is the entry function for the native operating system. This is where you should put any of your operating system specific code. For

instance, if you want to add a signal handler on Linux, you could do it here before calling OS_Main().

- **__os_init_lynxos.c–** This function is the entry function for the native operating system. This is where you should put any of your operating system specific code. For instance, if you want to add a signal handler on Lynxos, you could do it here before calling OS_Main().
- **__os_init_mqx.c–**This function is the entry function for the native operating system. This is where you should put any of your operating system specific code. For instance, if you wanted to add a signal handler on MQX, you could do it here before calling OS_Main().
- **__os_init_nucleus.c–** This function is the entry function for the native operating system. This is where you should put any of your operating system specific code. For instance, if you wanted to add a signal handler on Nucleus, you could do it here before calling OS_Main().
- **__os_init_qnx.c–** This function is the entry function for the native operating system. This is where you should put any of your operating system specific code. For instance, if you wanted to add a signal handler on QNX, you could do it here before calling OS_Main().
- **__os_init_solaris.c–** This function is the entry function for the native operating system. This is where you should put any of your operating system specific code. For instance, if you want to add a signal handler on Solaris, you could do it here before calling OS_Main().
- **__os_init_threadx.c–** This function is the entry function for the native operating system. This is where you should put any of your operating system specific code. For instance, if you want to add a signal handler on ThreadX, you could do it here before calling OS_Main().
- **__os_init_uitron.c–** This function is the entry function for the native operating system. This is where you should put any of your operating system specific code. For instance, if you want to add a signal handler on micro-ITRON, you could do it here before calling OS_Main().
- **__os_init_vxworks.c–** This function is the entry function for the native operating system. This is where you should put any of your operating system specific code. For instance, if you want to add a signal handler on VxWorks, you could do it here before calling OS_Main().
- **__os_init_windows_host.c–**These functions are the various entry functions for the different operating systems. This is where you should put any of your operating system specific code. For instance, if you want to add a signal handler on Windows host, you could do it here before calling OS_Main().When optimizing, you will need to write an equivalent function for your target operating system.
- **__os_init_windows.c–** These functions are the various entry functions for the different operating systems. This is where you should put any of your operating system specific code. For instance, if you want to add a signal handler on Windows, you could do it here before calling OS_Main(). When optimizing, you will need to write an equivalent function for your target operating system**.**
- **__os_init_android.c–**This function is the entry function for the native operating system. This is where you should put any of your operating system specific code. For instance, if you want to add a signal handler on android, you could do it here before calling OS_Main().
- **__os_init_ucos.c–**This function is the entry function for the native operating system. This is where you should put any of your operating system specific code. For

instance, if you want to add a signal handler on uCOS, you could do it here before calling OS_Main().

- **__os_init_netbsd.c–**This function is the entry function for the native operating system. This is where you should put any of your operating system specific code. For instance, if you want to add a signal handler on NetBSD, you could do it here before calling OS_Main().
- **os_application_start.c–** This function is the first OS agnostic function and should be the start point for the application development**.**
- **os_library_init.c–** This function initializes the required Interface products and creates the entry threads for each product.
- **Os_main.c–** This function initializes the Cross-OS Interface layer and calls OS_Application_Wait_For_End which will suspend until OS_Application_Free or OS_Delete_Process is called. It also spawns the first OS Independents thread which is the true entry point for Cross-OS Interface.

The application code starts in the os_library_init.c file, the user defined entry function and name of the application for Cross-OS Interface can be specified in:

```
#define OS_ABSTRACTOR_BASE_ENTRY_FUNCTION
#define OS_APPLICATION_START_TASK_NAME
```

For Vxworksinterface, the user defined entry function and stack size can be specified in:

```
#define VXWORKS_ENTRY_FUNCTION
#define VXWORKS_ENTRY_FUNCTION_STACK_SIZE
```

Similarly, this is how it works for all the remaining changers/abstractors.

You can insert code that is only included when they use OS PAL host in the following way:

- For windows host you can insert code in __os_init_windows_host.c (inside main function before calling OS_MAIN), that is only included in OS PAL windows host.
- For Linux host, you can insert code in __os_init_linux_host.c (inside main function before calling OS_MAIN), that is only included in OS PAL Linux host.

You can insert code that is specific to a target OS (inside main function before calling OS_MAIN) in the following way:

- For LynxOS target, insert in __os_init_lynxos.c
- For mqx target, insert in __os_init_mqx.c
- For Linux target, insert in __os_init_linux.c
- For Nucleus target, insert in __os_init_nucleus.c
- For QNX target, insert in __os_init_qnx.c
- For Solaris target, insert in __os_init_solaris.c
- For Threadx target, insert in __os_init_threadx.c
- For uITRON target, insert in __os_init_uitron.c
- For VxWorks target, insert in __os_init_vxworks.c
- For Android target, insert in __os_init_android.c
- For uCOS target, insert in __os_init_ucos.c
- For NetBSD target, insert in __os_init_netbsd.c

## Host System Configuration

The below defines are the system settings used by the OS_Application_Init() function. Use these to modify the settings when running on the host. A value of -1 for any of these will use the default values located in cross_os_usr.h. When you optimize for the target side code, the wizard will create a custom cross_os_usr.h using the settings you specify at that time so these defines will no longer be necessary.

```
#define HOST_DEBUG_INFO                 -1
#define HOST_TASK_POOL_TIMESLICE        -1
#define HOST_TASK_POOL_TIMEOUT          -1
#define HOST_ROOT_PROCESS_PREEMPT       -1
#define HOST_ROOT_PROCESS_PRIORITY      -1
#define HOST_ROOT_PROCESS_STACK_SIZE    -1
#define HOST_ROOT_PROCESS_HEAP_SIZE     -1
#define HOST_DEFAULT_TIMESLICE          -1
#define HOST_MAX_TASKS                  -1
#define HOST_MAX_TIMERS                 -1
#define HOST_MAX_MUTEXES                -1
#define HOST_MAX_PIPES                  -1
#define HOST_MAX_PROCESSES              -1
#define HOST_MAX_QUEUES                 -1
#define HOST_MAX_PARTITION_MEM_POOLS    -1
#define HOST_MAX_DYNAMIC_MEM_POOLS      -1
#define HOST_MAX_EVENT_GROUPS           -1
#define HOST_MAX_SEMAPHORES             -1
#define HOST_USER_SHARED_REGION1_SIZE   -1
```

**OS_HOST**: This flag is used only in OS PAL environment. It is not used in the target environment.

Host mode defines can be modified in os_main.c file. For example, modify maximum tasks under host environment in HOST_MAX_TASKS.

**NOTE**: You can manually change the values in the Optimized Target Code GeneratorWizard. Refer to Generating Optimized Target Code chapter in the manual.

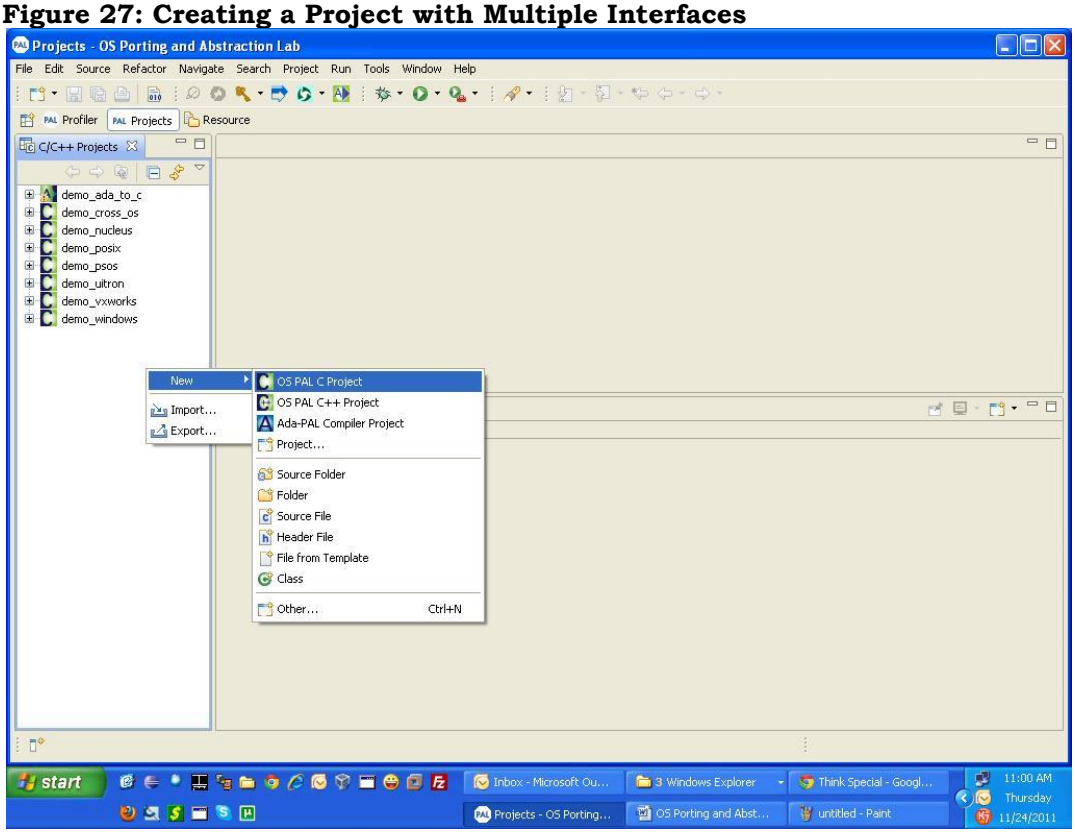
# Creating a Project with Multiple Interfaces

**NOTE**: This feature requires a license. Click http://mapusoft.com/downloads/ospal-evaluation/ to request an evaluation license.

To create a project with multiple interfaces:

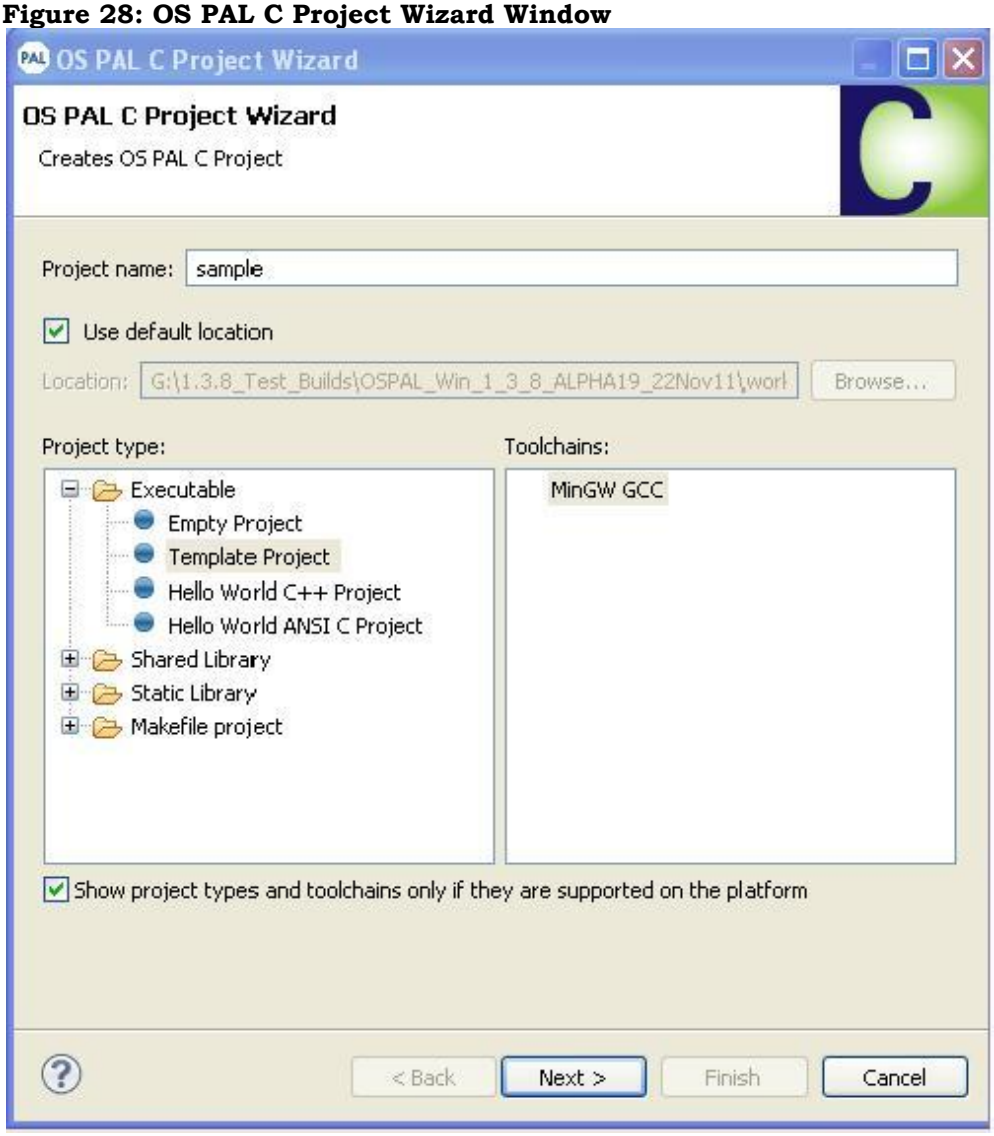
1. From OS PAL main window, select any project under C/C++ **Projects** tab on the left pane.
2. Select **New > OS PAL C/C++ Project** as shown in Figure 27.

**Figure 27: Creating a Project with Multiple Interfaces**



3. On OS PAL C Project Wizard window, type a project name and give a location next to **Project Name** text box.
4. Under Project Types, expand the **Executable** menu. Select **Template Project** and click **Next** as shown in Figure 28.

**Figure 28: OS PAL C Project Wizard Window**

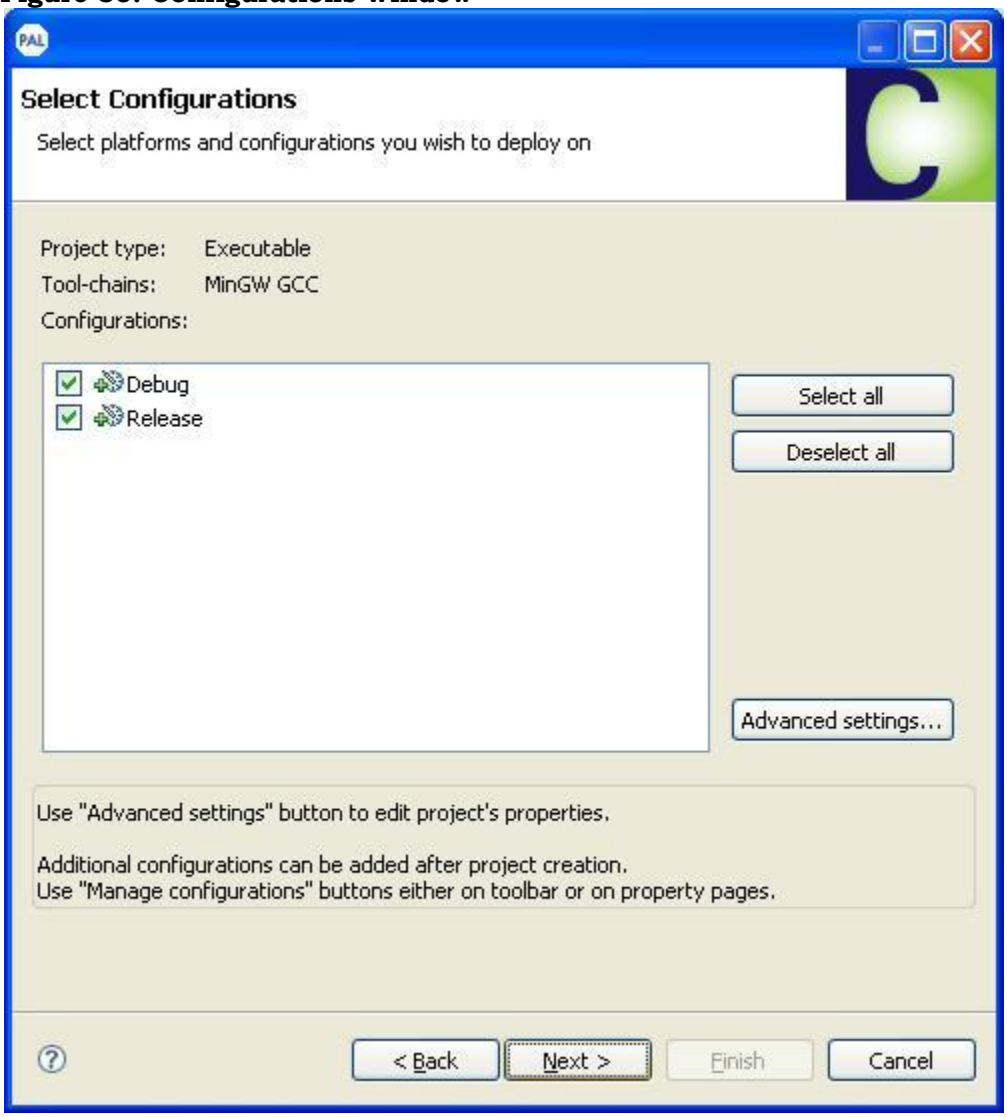


5. On Basic Settings window, define the basic properties of your project and click **Next** as shown inFigure 29.

**Figure 29: Basic Settings Window**

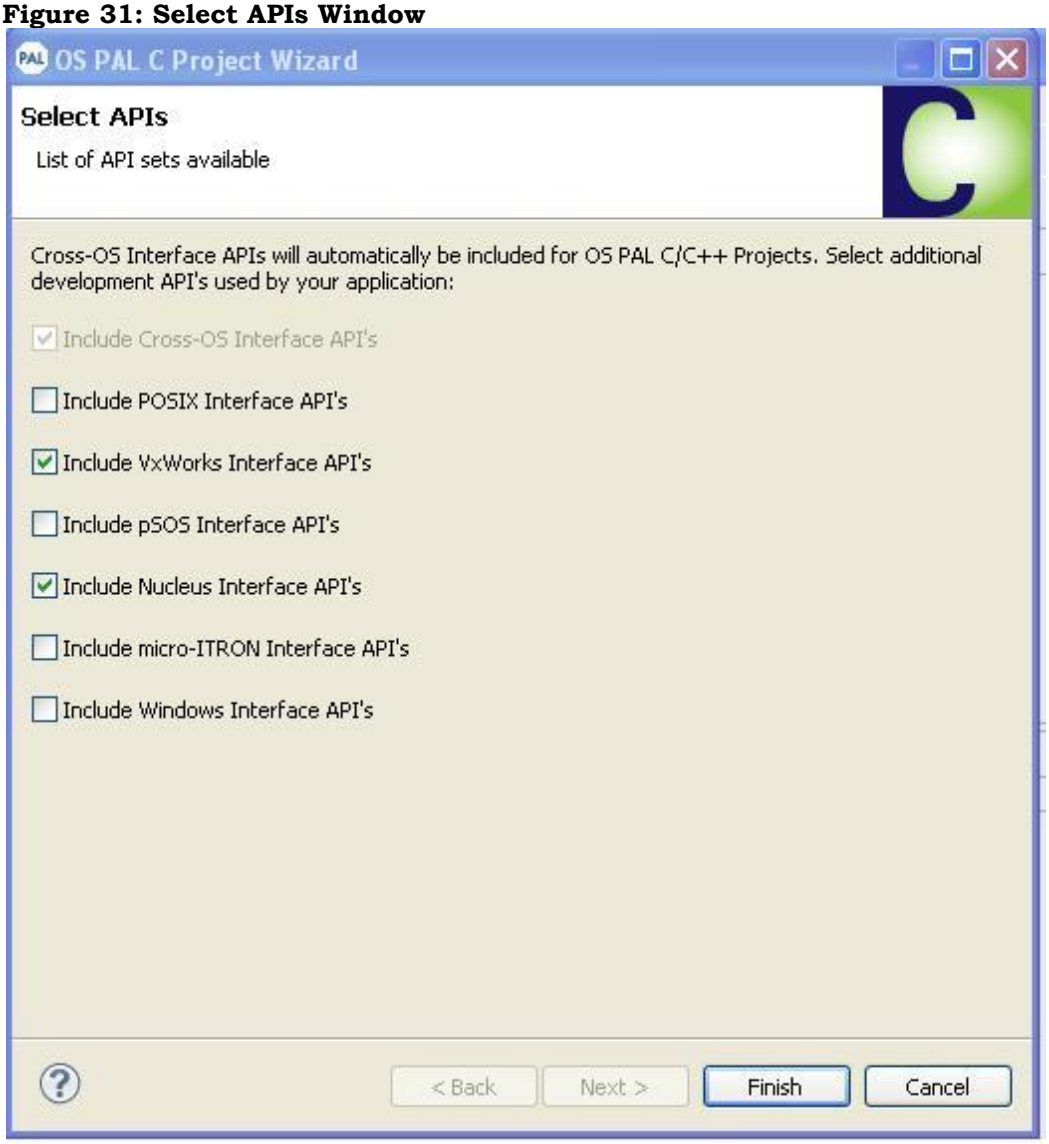
6. On Select Configurations window, select the platforms and configurations for deployment and click **Next** as shown inFigure 30.

**Figure 30: Configurations Window**



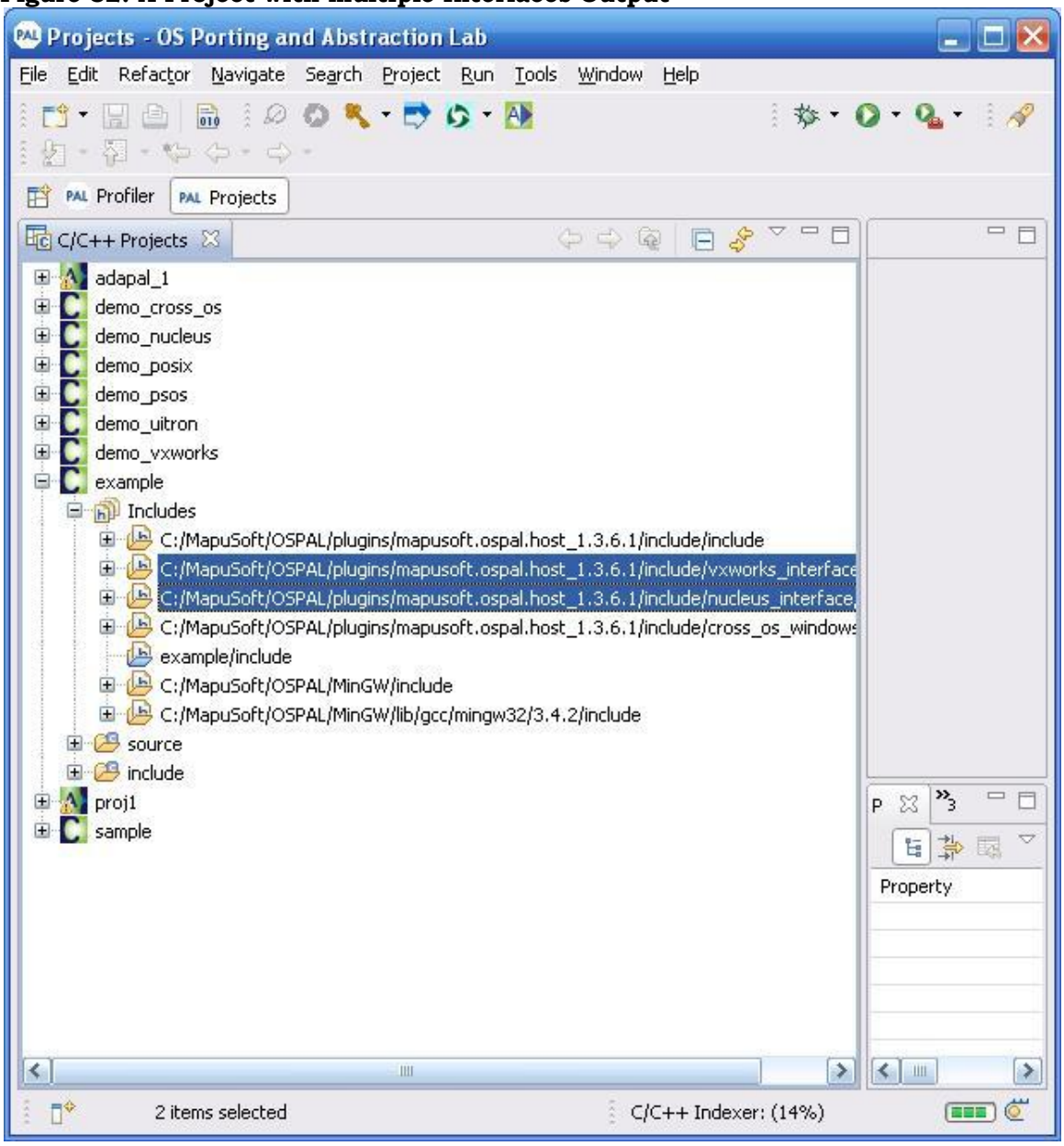
7. On Select APIs window, select the check box before **Include Nucleus and VxWorks Interfaces APIs**, and click **Finish** as shown in Figure 31.

**Figure 31: Select APIs Window**



You will see the output as shown in Figure 32.

**Figure 32: A Project with multiple Interfaces Output**



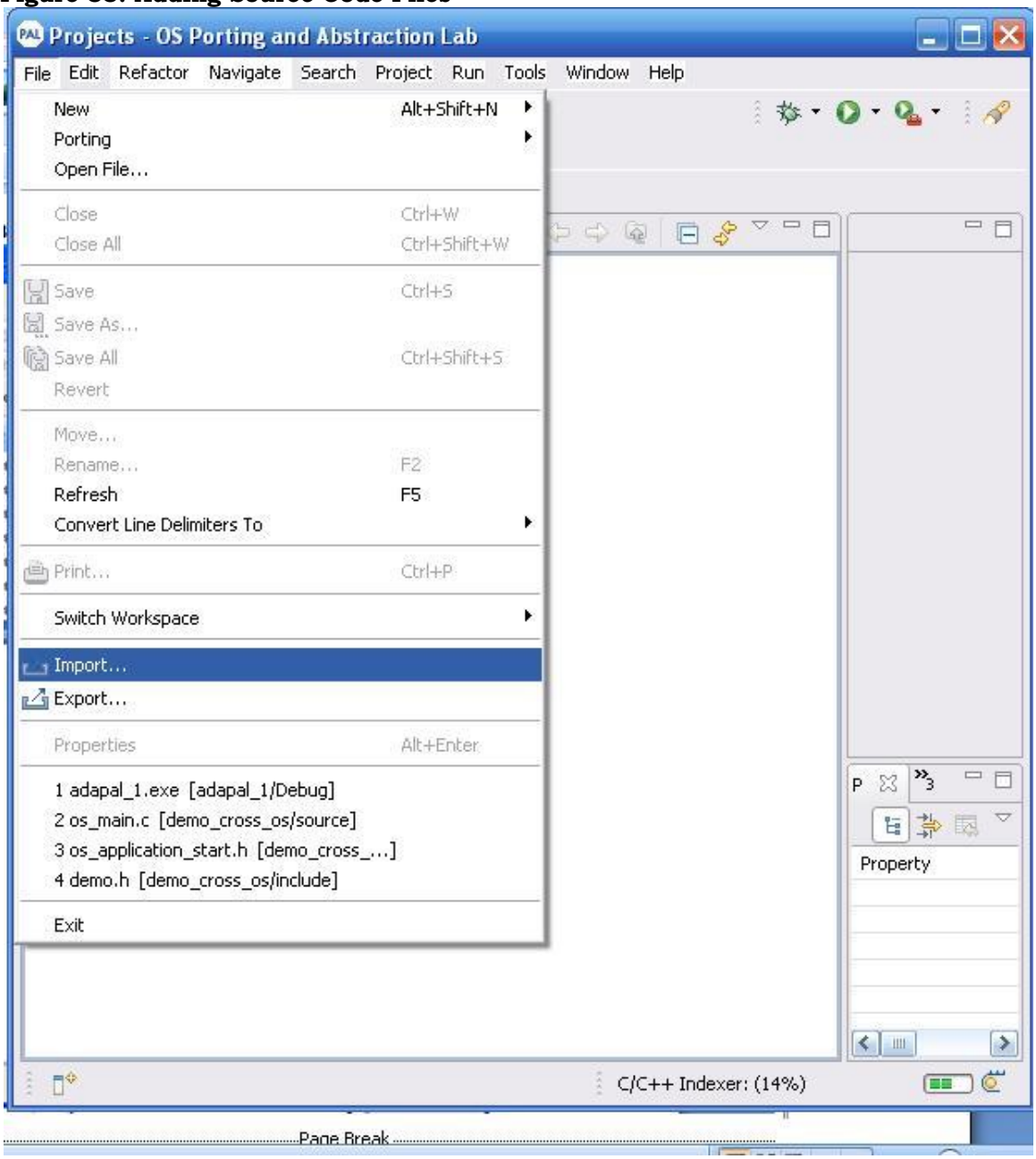
## Adding Source Code Files to OS PAL Project

**NOTE**: This feature requires a license. Click http://mapusoft.com/downloads/ospal-evaluation/ to request an evaluation license.

To add source code files:

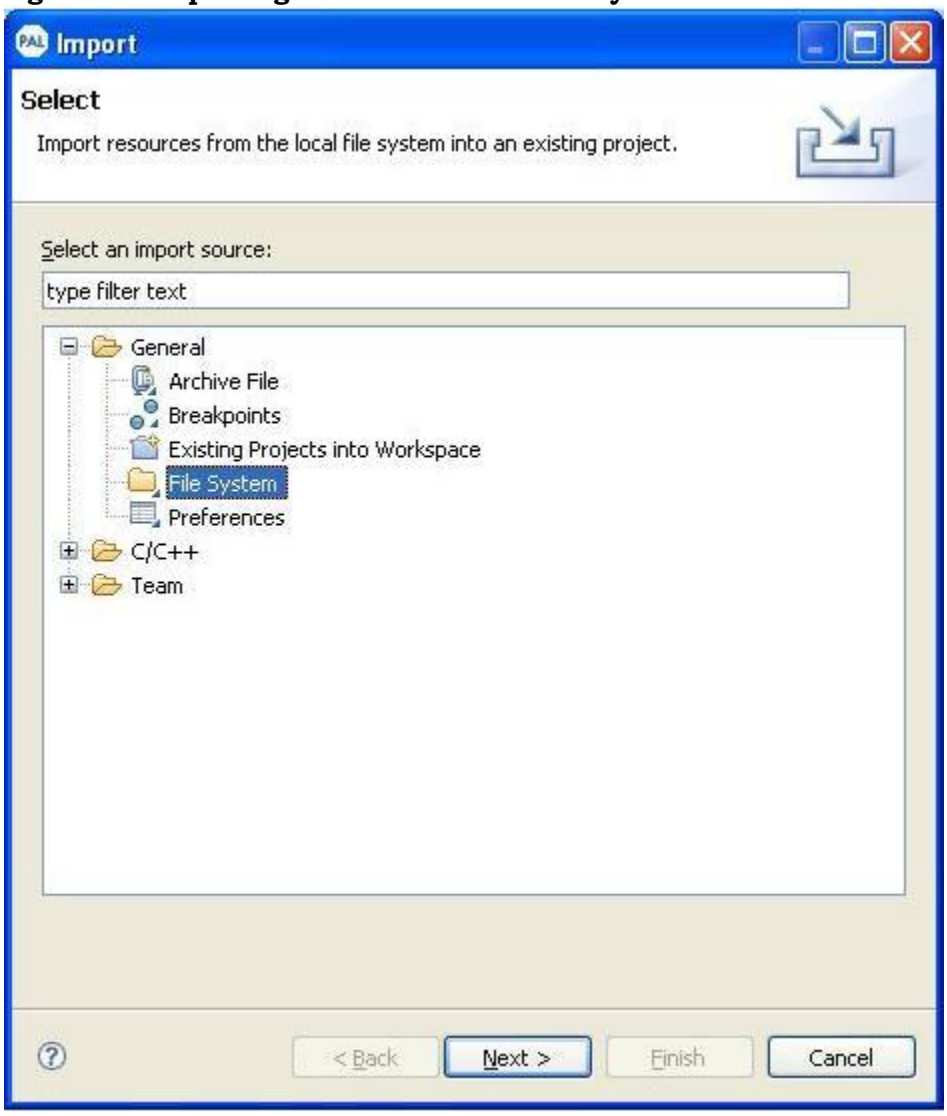
1. Select a project under C/C++ Projects pane.
2. Right click on it and select **Import** as shown in Figure 33.

**Figure 33: Adding Source Code Files**



3. Select your import source and click **Next** as shown in Figure 34.

**Figure 34: Importing Files from Local File System**

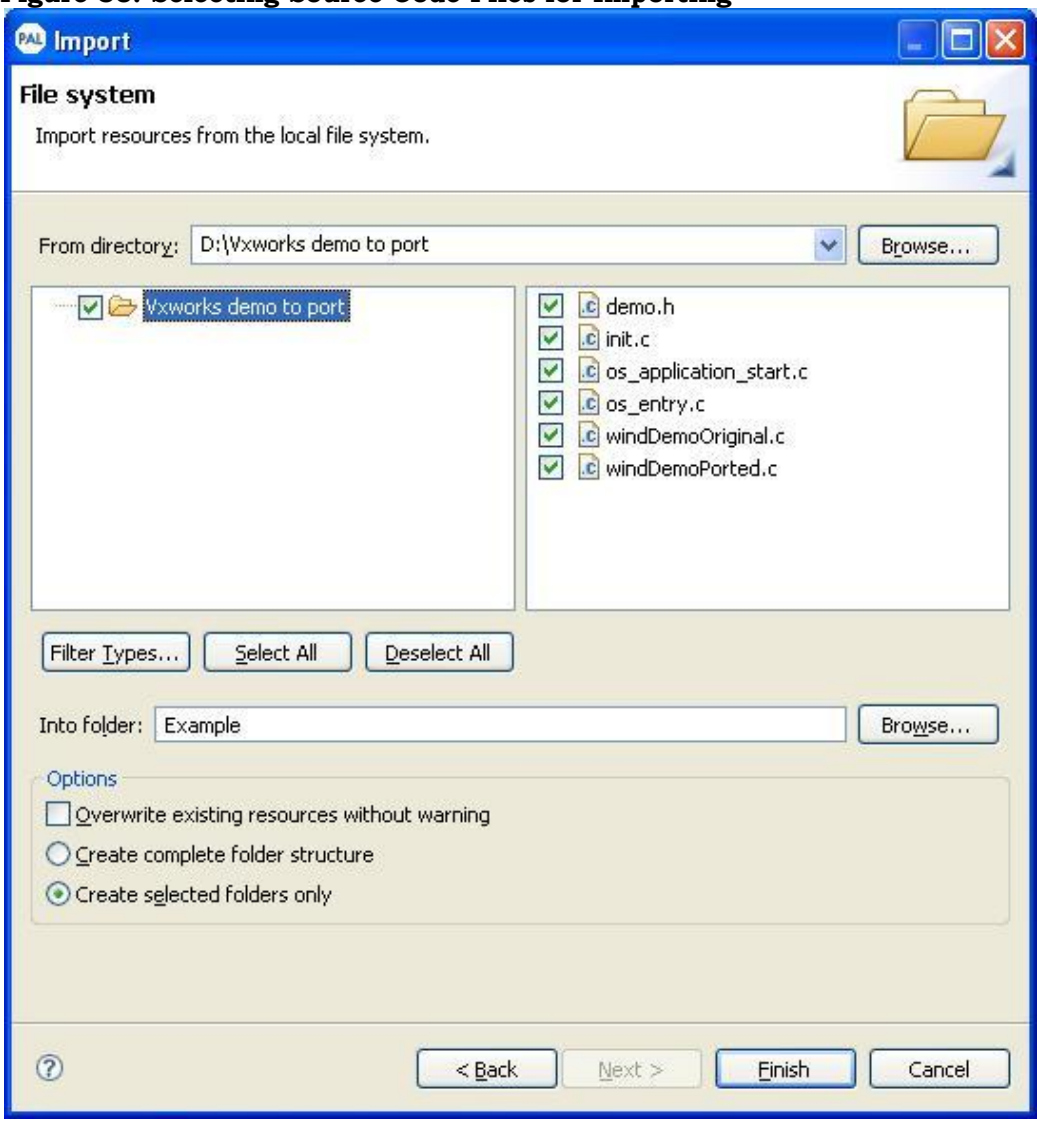


4. Select the directory on your local file system which contains the source code files and click **OK** as shown in Figure35.

**Figure35: Importing Source Code Files from Directory**

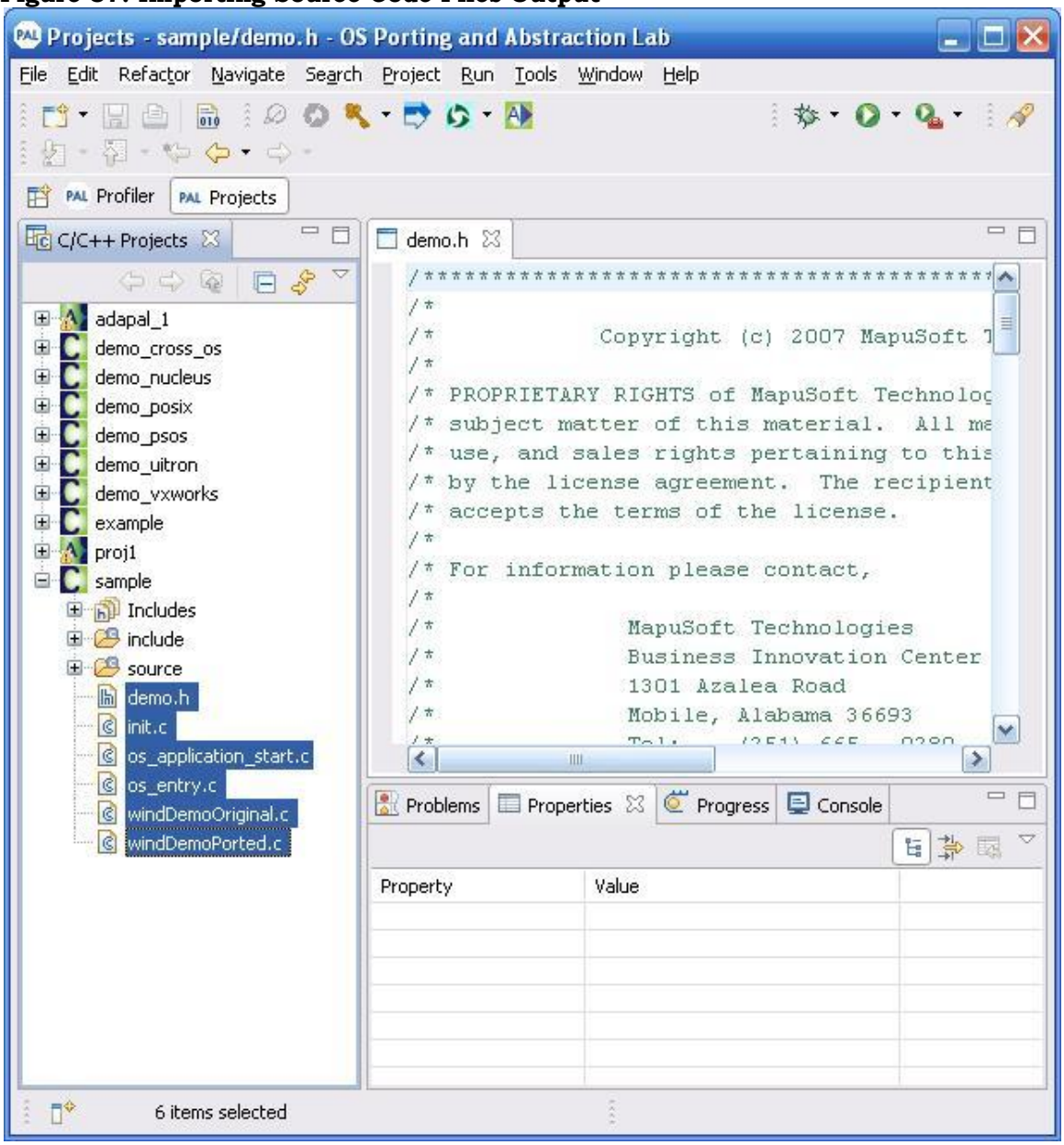
5. Select the check boxes corresponding to the source code files you want to import and click **Finish** as shown in Figure 36.

**Figure 36: Selecting Source Code Files for Importing**



6. You can view the source code files added to your OS PAL project as shown in Figure 37.

**Figure 37: Importing Source Code Files Output**

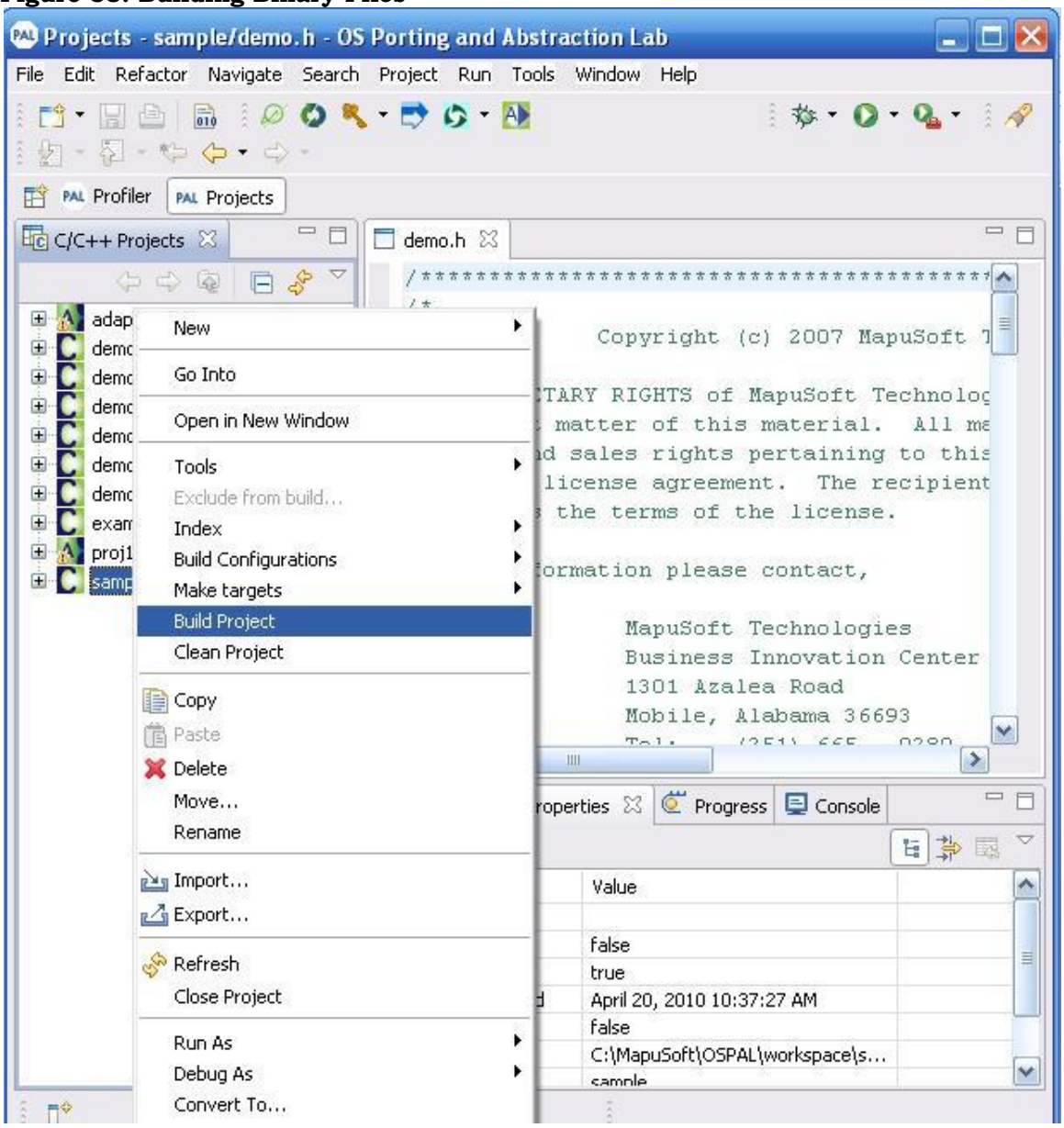


## Building Binary Files for a Project

**NOTE**: This feature requires a license. Click http://mapusoft.com/downloads/ospal-evaluation/ to request an evaluation license.

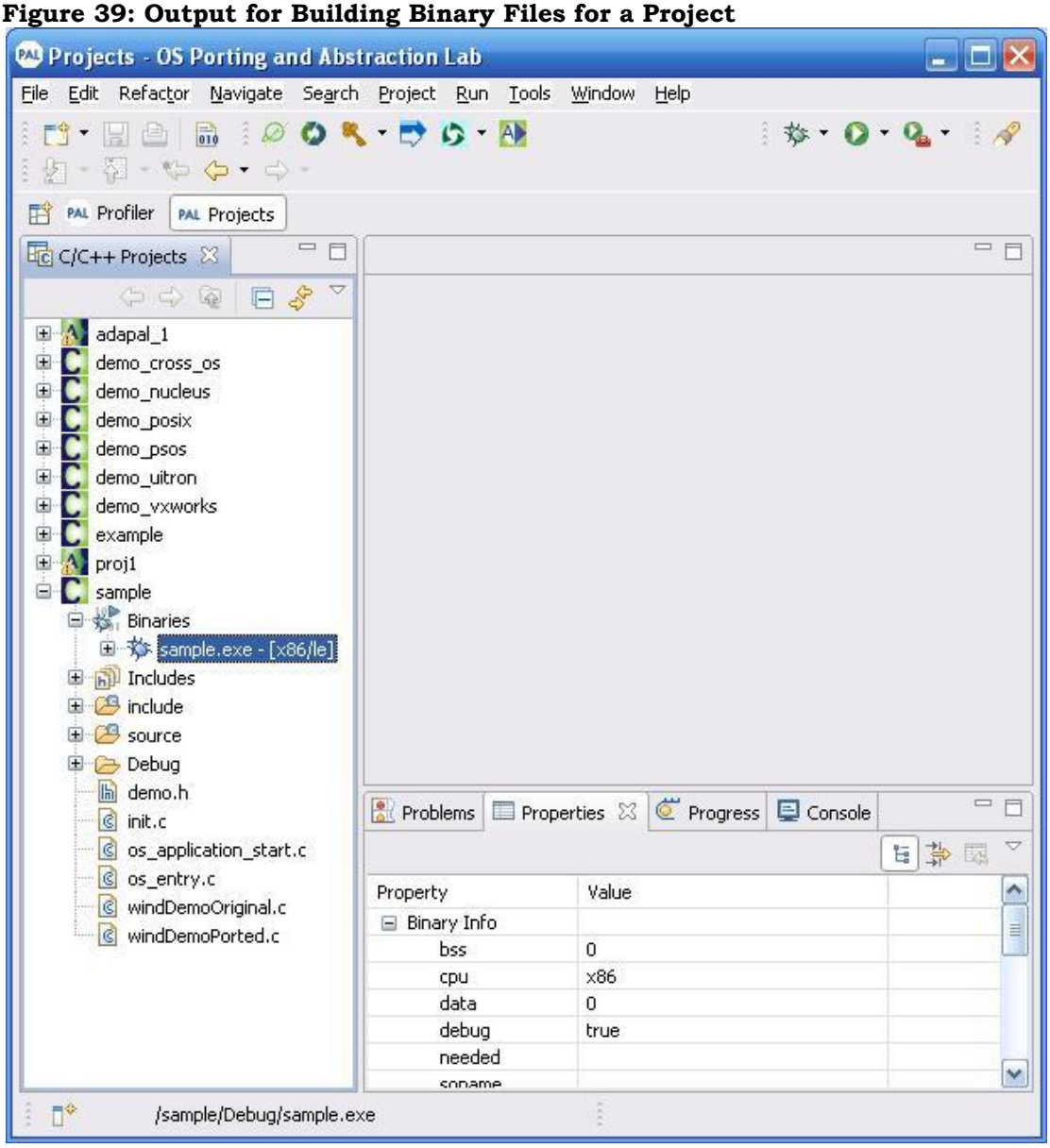
1. Creating an OS PAL C/C++ Project.
2. Adding Source Code Files to OS PAL Project.
3. Select a project under C/C++ Projects pane, right click and select **Build Project** as shown in Figure 38.

**Figure 38: Building Binary Files**



4. You can view how the binary files are built in Figure 39.

**Figure 39: Output for Building Binary Files for a Project**

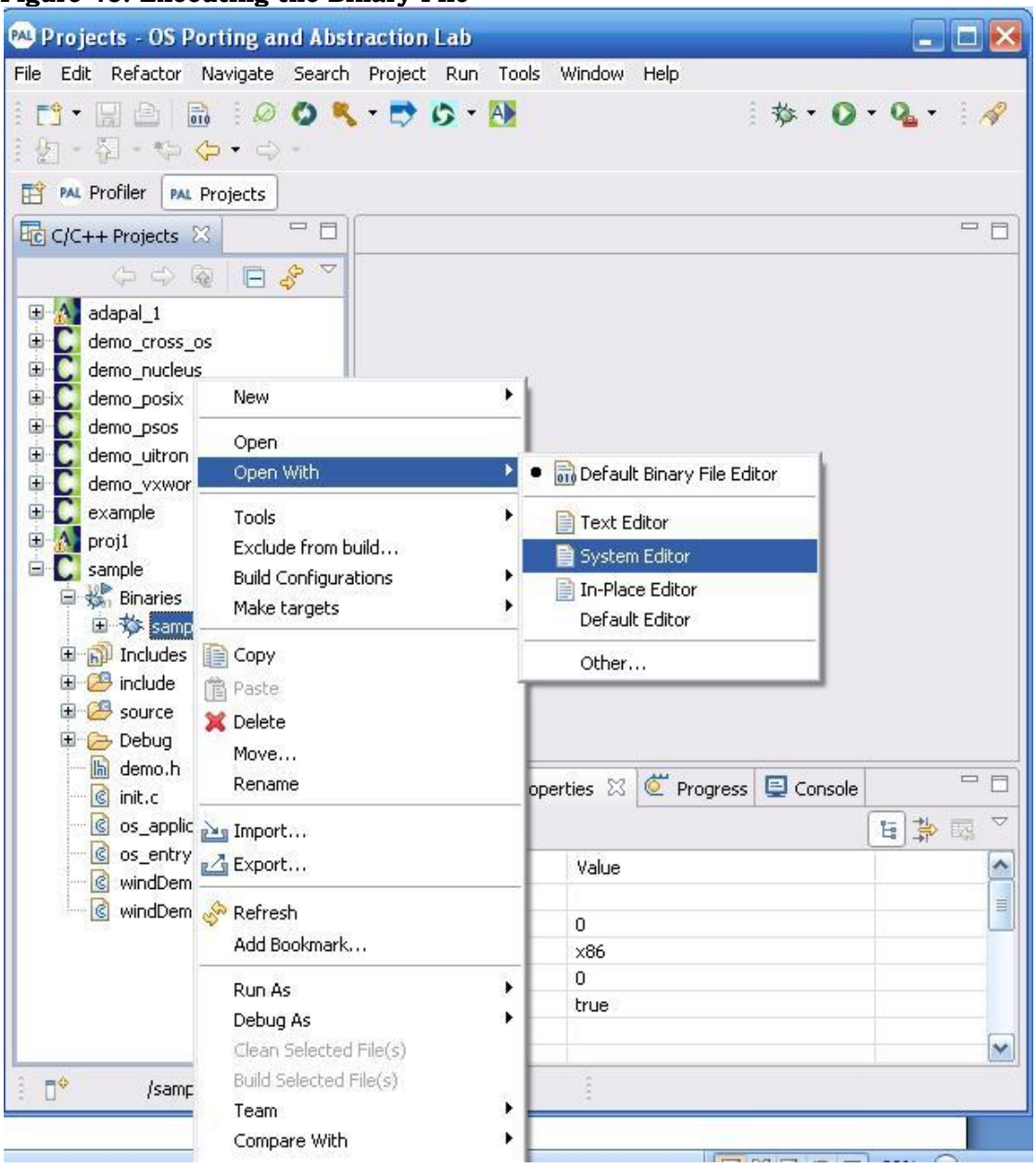


## Executing Binary Files

To execute the binary in Windows host:

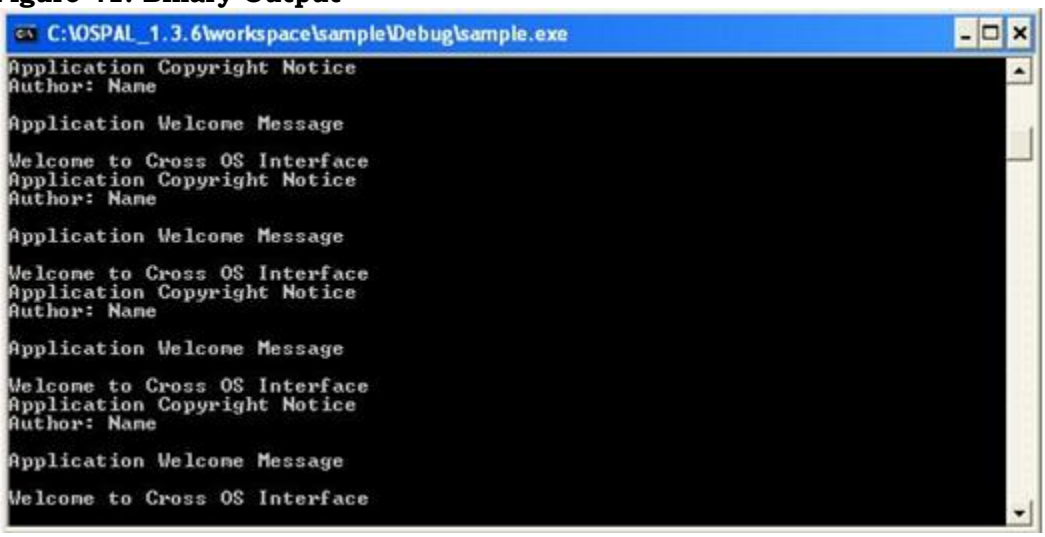
1. Select Project that you have created.
2. Select the Binary file, right click and select **Open With > System Editor** as shown in Figure 40.

**Figure 40: Executing the Binary File**



3. This approach always will fork a terminal to view the output as shown in Figure 41.

**Figure 41: Binary Output**

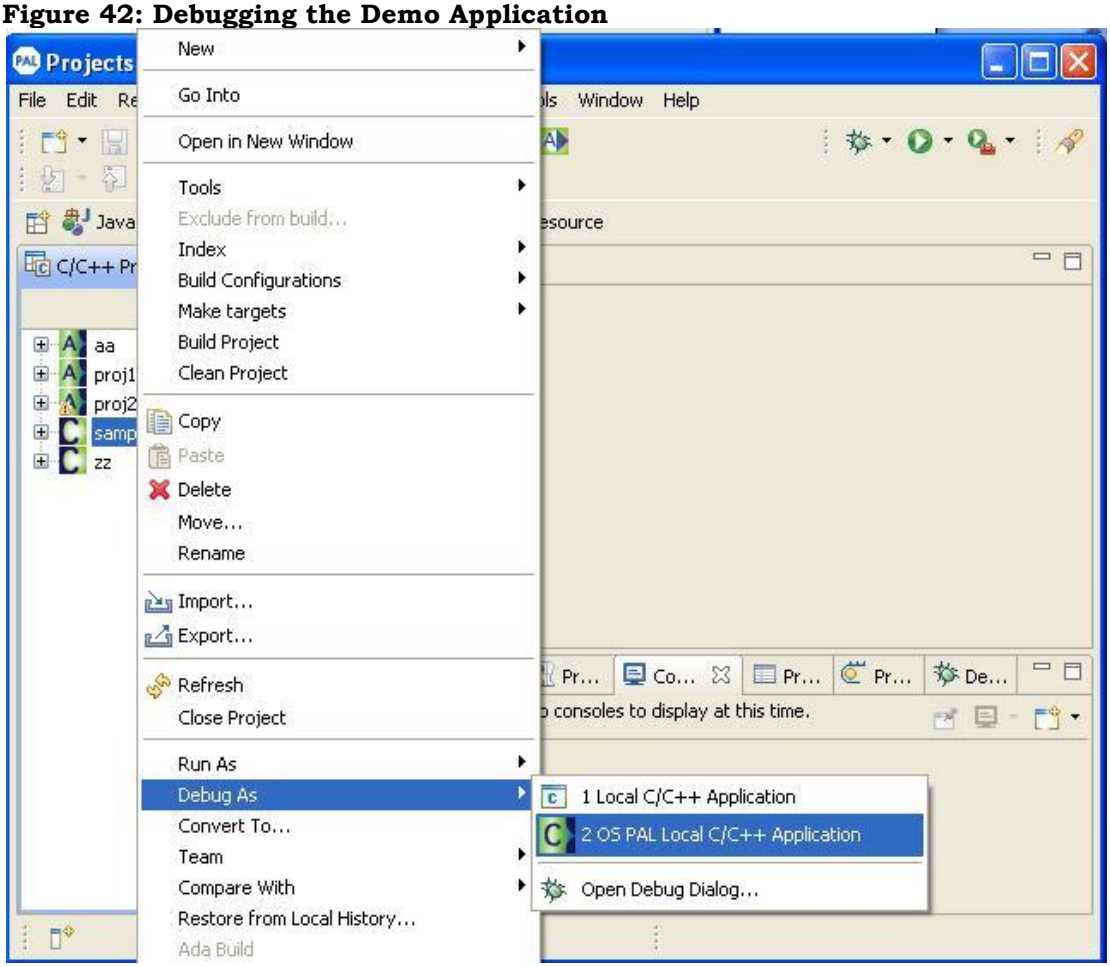


## Debugging the Demos Supplied by MapuSoft

**Example**: Debugging the demo_cross_os application

1. From OS PAL main window, select demo_cross_os project.
2. Right click on the project and select **Debug as > Local OS PAL C\C ++ Application** or click on debugging icon as highlighted in Figure 42.

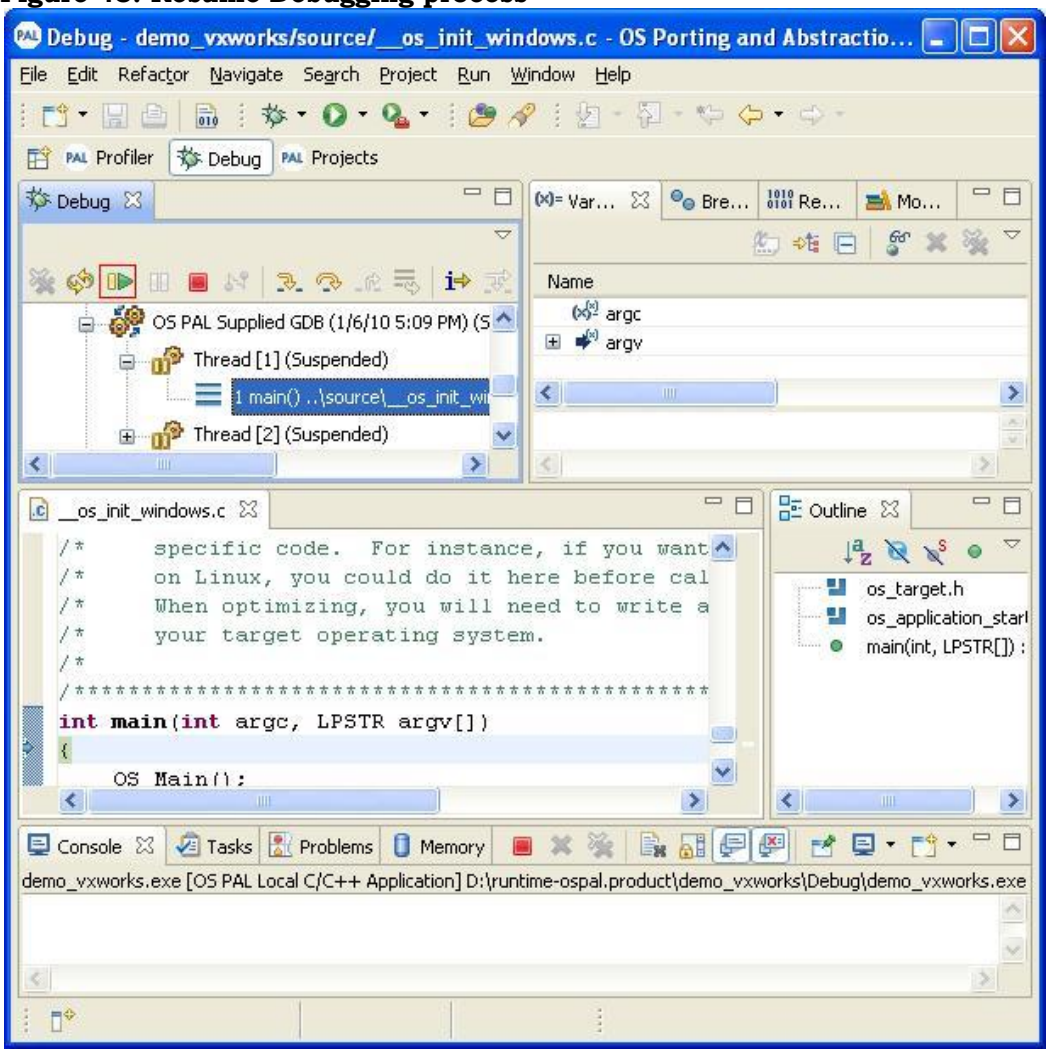
**Figure 42: Debugging the Demo Application**




**NOTE**: If the user uses the Debug dialog to create a new configuration then they need to select **OS PAL Local C/C++ Application** before creating. The other option is to not use the debug dialog, but instead select "OS PAL Local C/C++ Application" from the Debug As... menu. This method will create the correct configuration automatically.

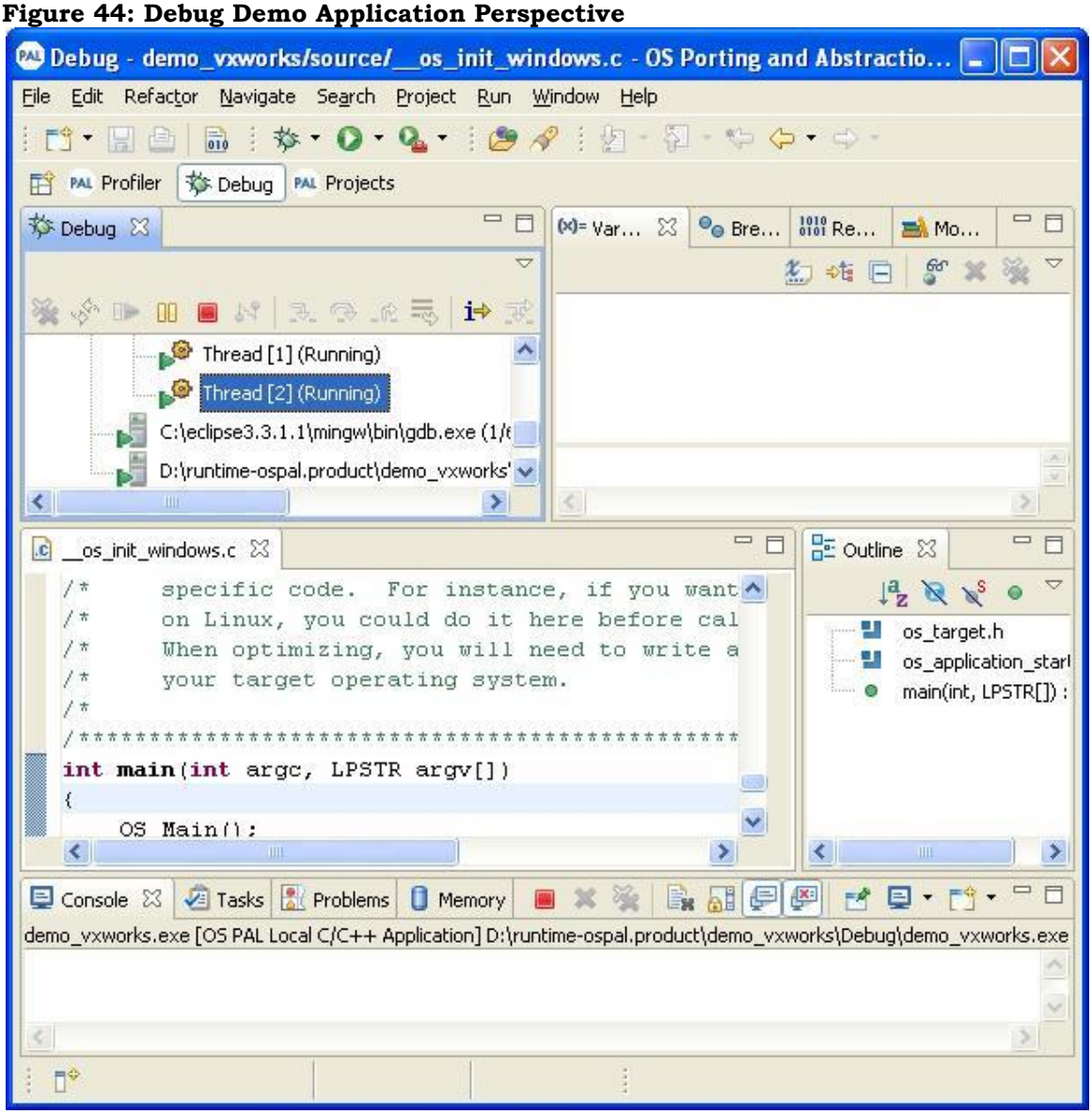
3. Debugging stops at the main function. Click **Resume** icon to resume the debugging process as shown in Figure 43.

**Figure 43: Resume Debugging process**



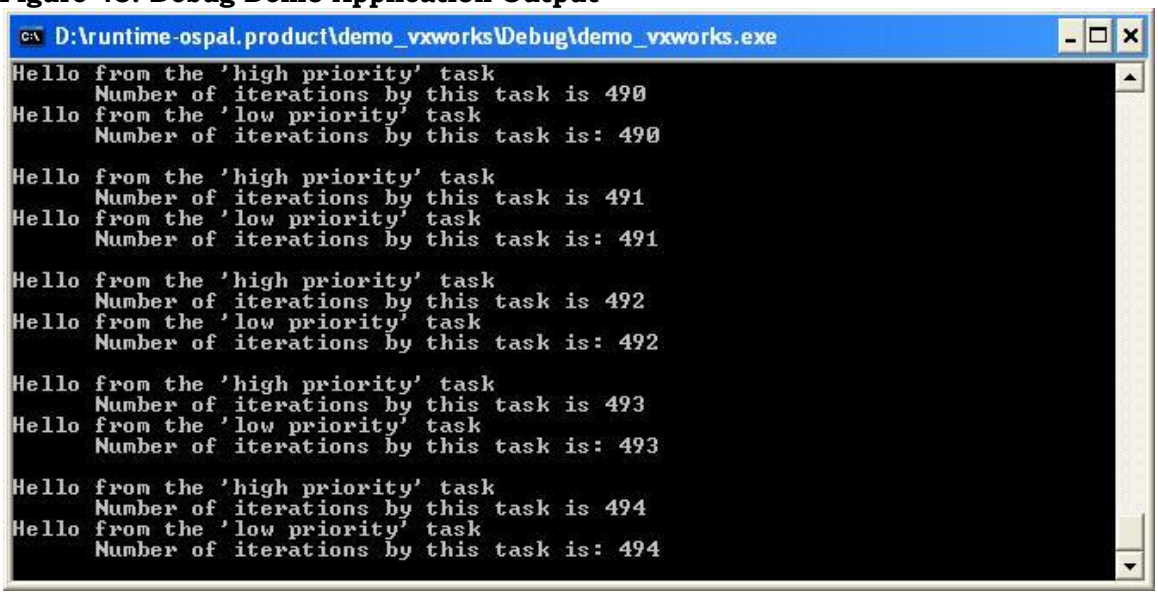
4. The debugging resumes as shown in Figure 44.

**Figure 44: Debug Demo Application Perspective**



You can see the debugging on the console as shown in Figure 45.

**Figure 45: Debug Demo Application Output**

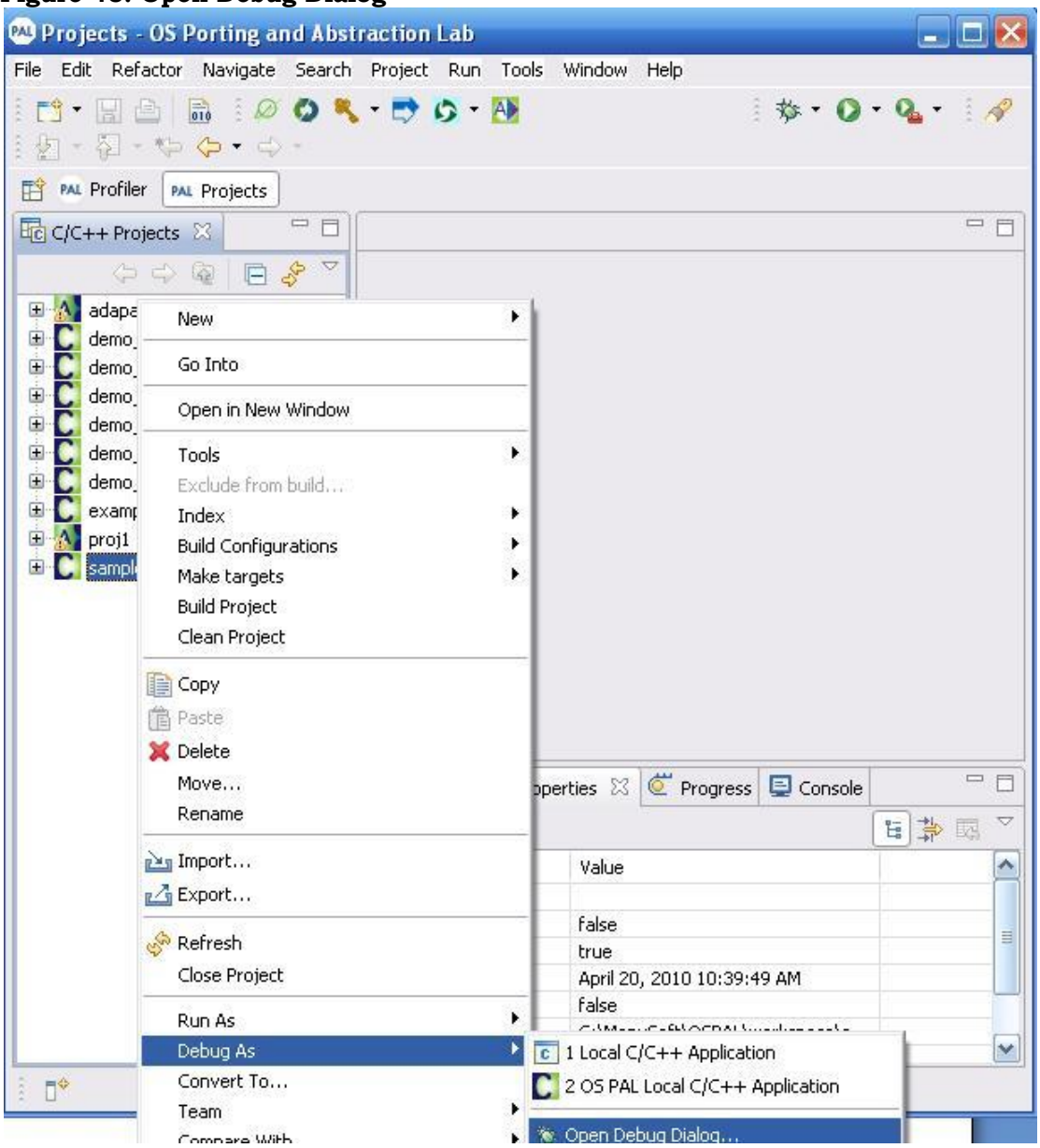


## Debugging Using External Console/Terminal

Debugging can be done using an external console or terminal in the following way:

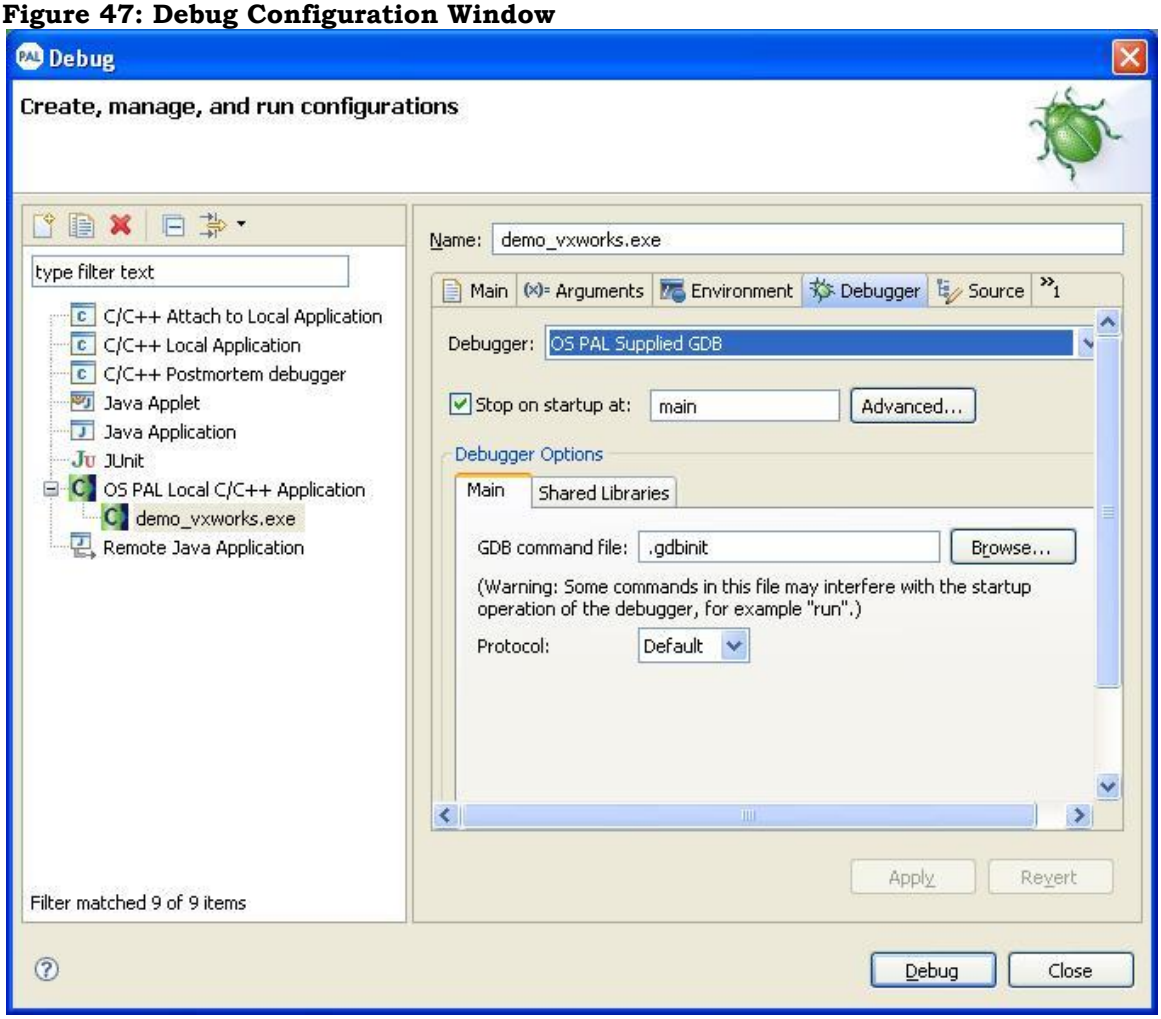
1. From OS PAL main window, select the demo_cross_os project.
2. Right click on the project and select **Debug as > Open Debug Dialog** as shown in Figure 46.

**Figure 46: Open Debug Dialog**



3. On Debug Configuration window, you can set your options for debugging as shown in the Figure 47. **NOTE**: You must use MapuSoft Supplied GDB to execute debugging.
   **NOTE:** OS PAL does not support Cygwin tools and its use is not recommended.

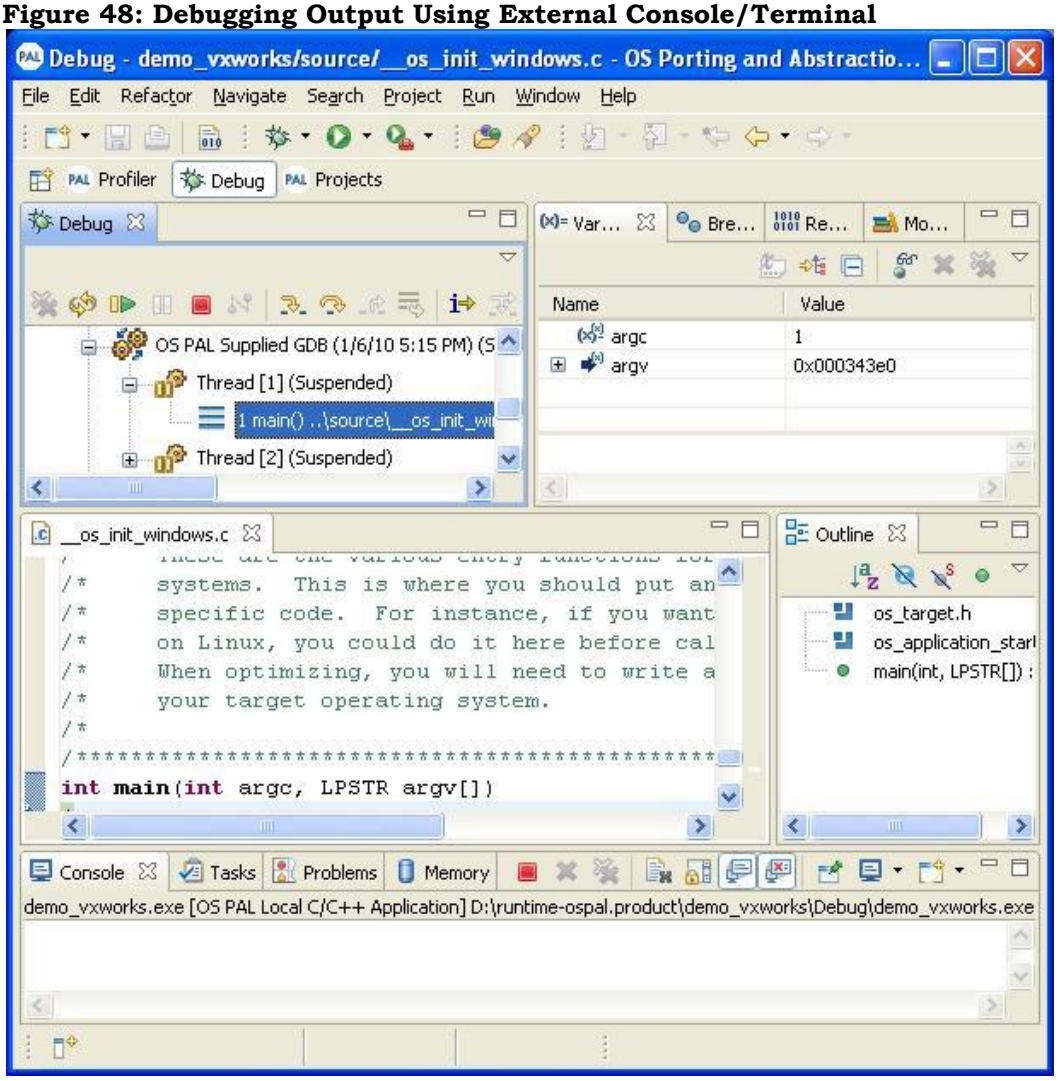
**Figure 47: Debug Configuration Window**




4. You can change any of the options here and click **Apply.**

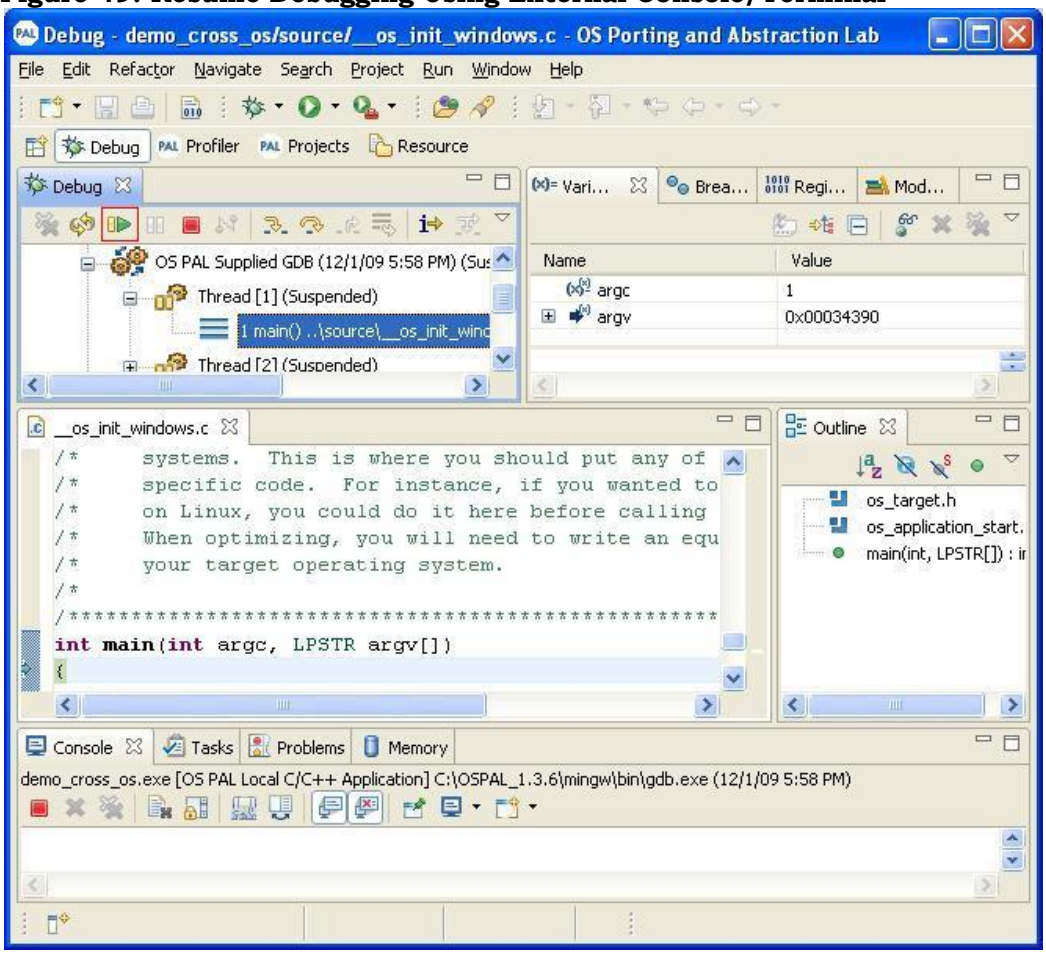
5. Click **Debug** to execute debugging using the external console or Terminal**.** You can view the debugging process in your console as shown in Figure 48.

**Figure 48: Debugging Output Using External Console/Terminal**



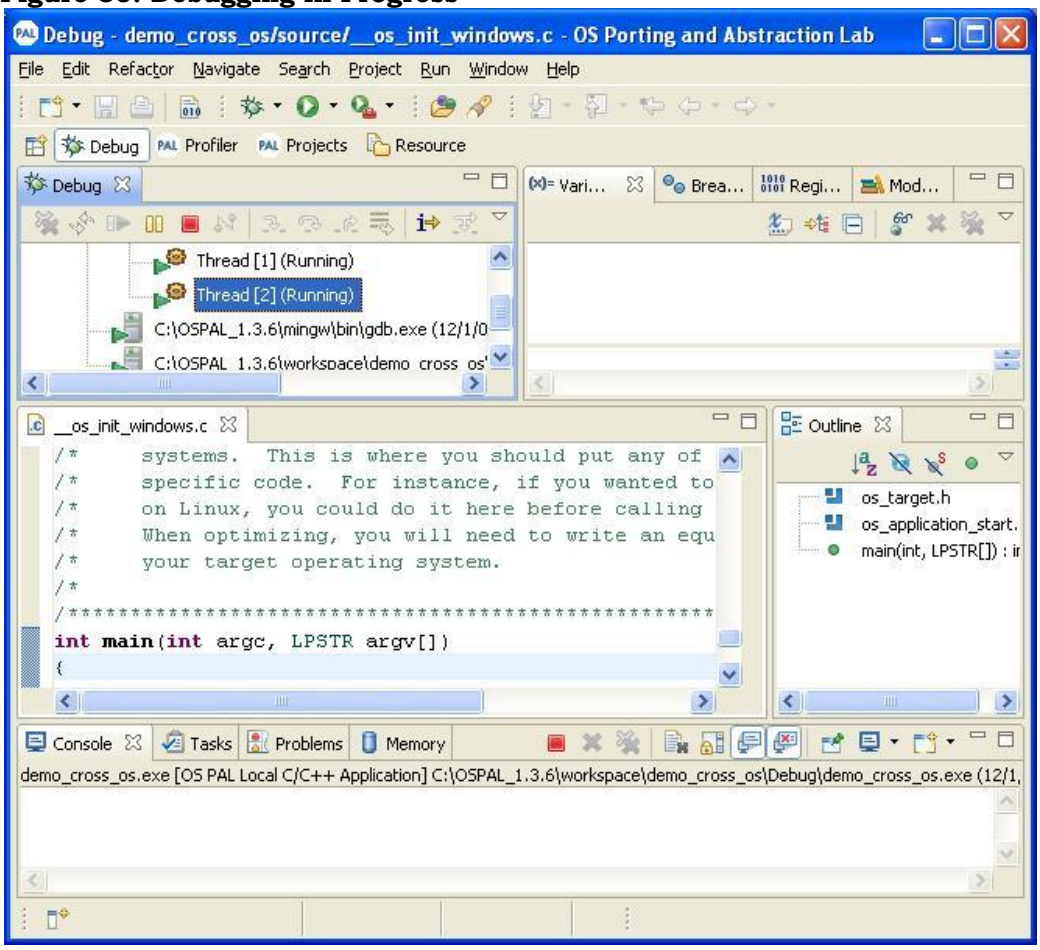
6. To resume debugging, click the resume icon  on the debugging window as shown in the Figure 49.

**Figure 49: Resume Debugging Using External Console/Terminal**



7. You have now successfully finished debugging by using external console or terminal as shown in Figure 50.

**Figure 50: Debugging in Progress**



# Inserting Application Code to Run only on Host Environment

The below defines are the system settings used by the OS_Application_Init() function. Use these to modify the settings when running on the host. A value of -1 for any of these will use the default values located in cross_os_usr.h.

When you optimize for the target side code, the wizard will create a custom cross_os_usr.h using the settings you specify at that time so these defines will no longer be necessary.

You can add some application code or debug statements like printf, assert, which is mostly used in host environment only. This line of code will be ignored by the compiler in target environment.

## OS HOST Selection

The flag has to be false for Full Source Library Package generation.

| Flag and Purpose | Available Options |
|---|---|
| **OS_HOST**<br>To select the host operating system | This flag is set as OS_TRUE by default in OS PAL environment. |

## Target 64 bit CPU Selection

Based on the OS you want the application to be built, set the following pre-processor definition in your project setting or make files:

| Flag and Purpose | Available Options |
|---|---|
| **OS_CPU_64BIT**<br>To select the target CPU type. | The value of OS_CPU_64BIT can be any ONE of the following:<br>• OS_TRUE – Target CPU is 64 bit type CPU<br>• OS_FALSE – Target CPU is 32 bit type CPU<br><br>**NOTE:** In New C projects creation, the flag OS_CPU_64BIT will be set to OS_FALSE by default and user needs to make this true if they run on a 64-bit OS. |

## Updating Project Settings

OS PAL provides exclusive way to update the Projects Settings by just a click of a button. This is very useful in any one of the following cases:

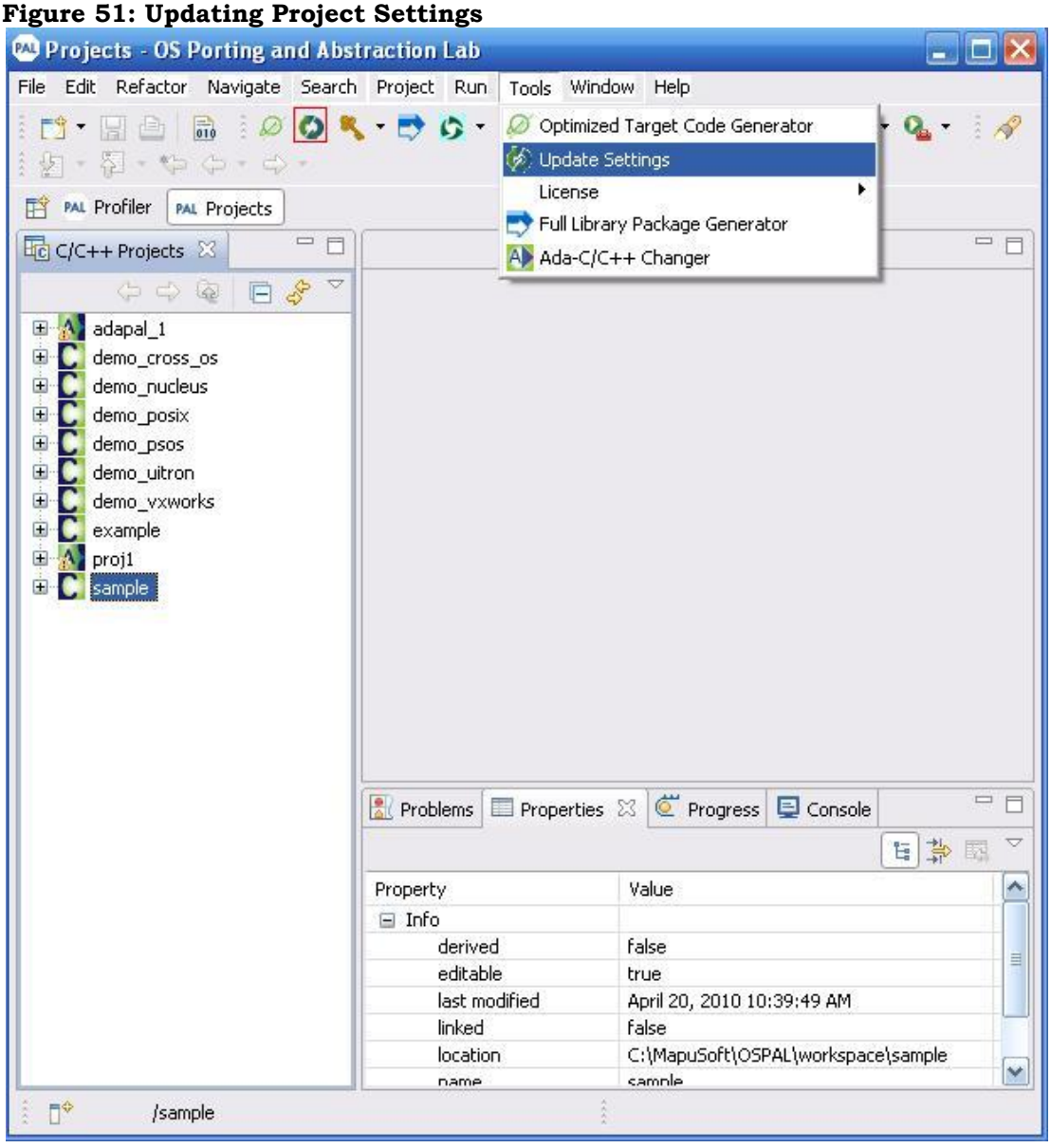
1. If the user has moved his workspace to a different location
2. If the project requires new toolchain that is installed recently

The **Update** button performs an auto update on all the projects updates which include files, new directory structures, libraries, and toolchains to the Project Settings.

To update project settings:

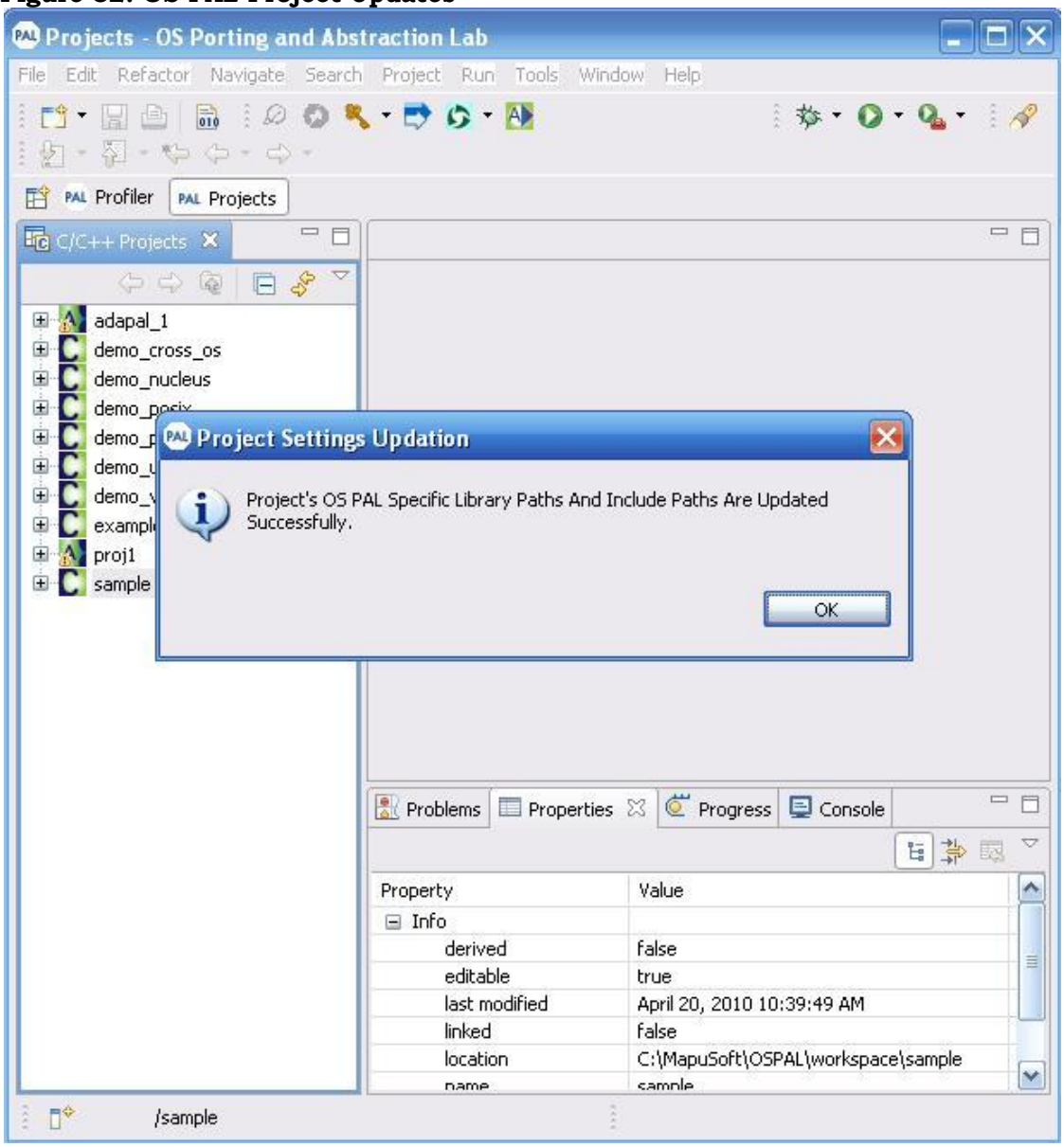
1. From OS PAL main menu select **Tools > Update Settings** or click OS PAL Project Settings button on OS PAL main menu or, as shown inFigure 51.

**Figure 51: Updating Project Settings**



2. OS PAL does an auto search for the project updates and updates the settings as shown in the Project Settings Updation window in Figure 52.

**Figure 52: OS PAL Project Updates**



# Porting VxWorks Applications using OSPAL

Porting applications into the OS PAL host environment can be done in three different ways:

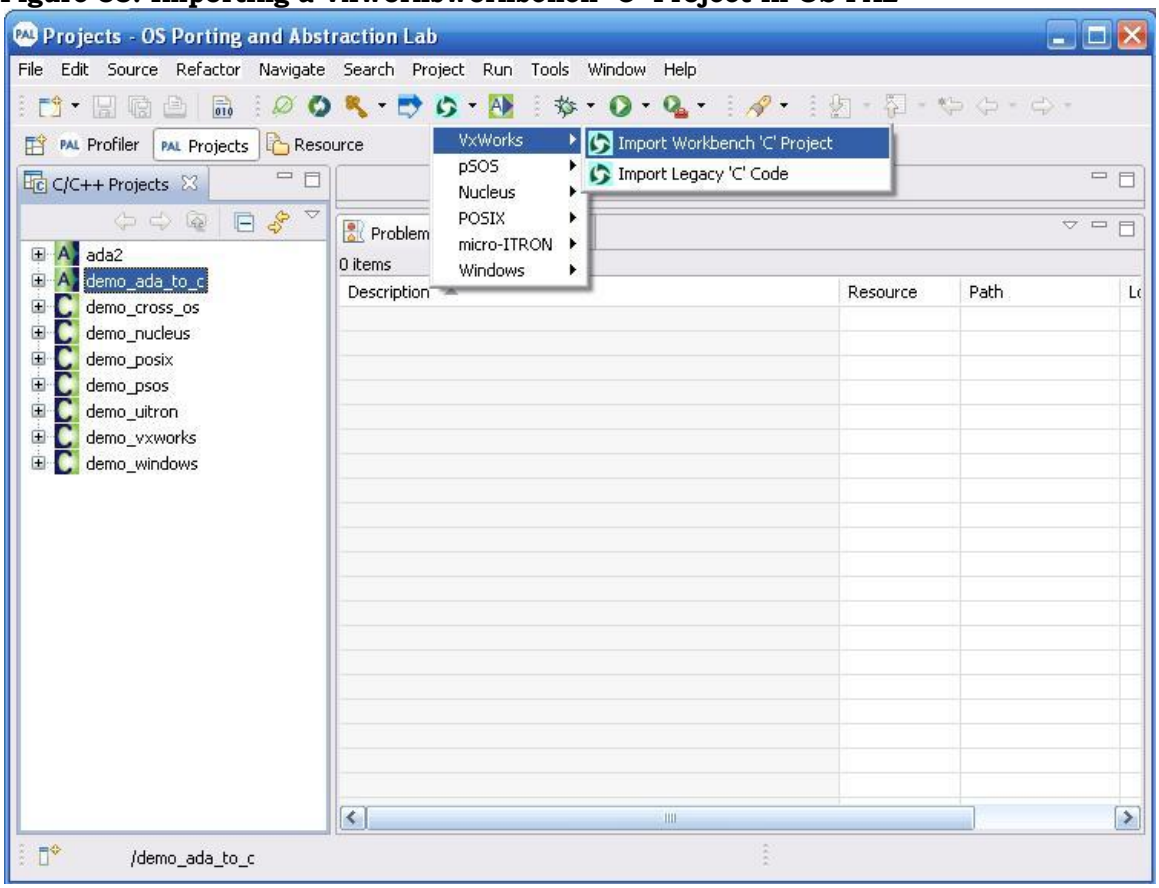
1. Porting a WindRiver Workbench project
2. Porting a Legacy application
3. Manual porting using OSPAL

## Method 1– Porting a WindRiver Workbench'C' Project

**NOTE**: This feature requires a license. Click http://mapusoft.com/downloads/ospal-evaluation/ to request an evaluation license.

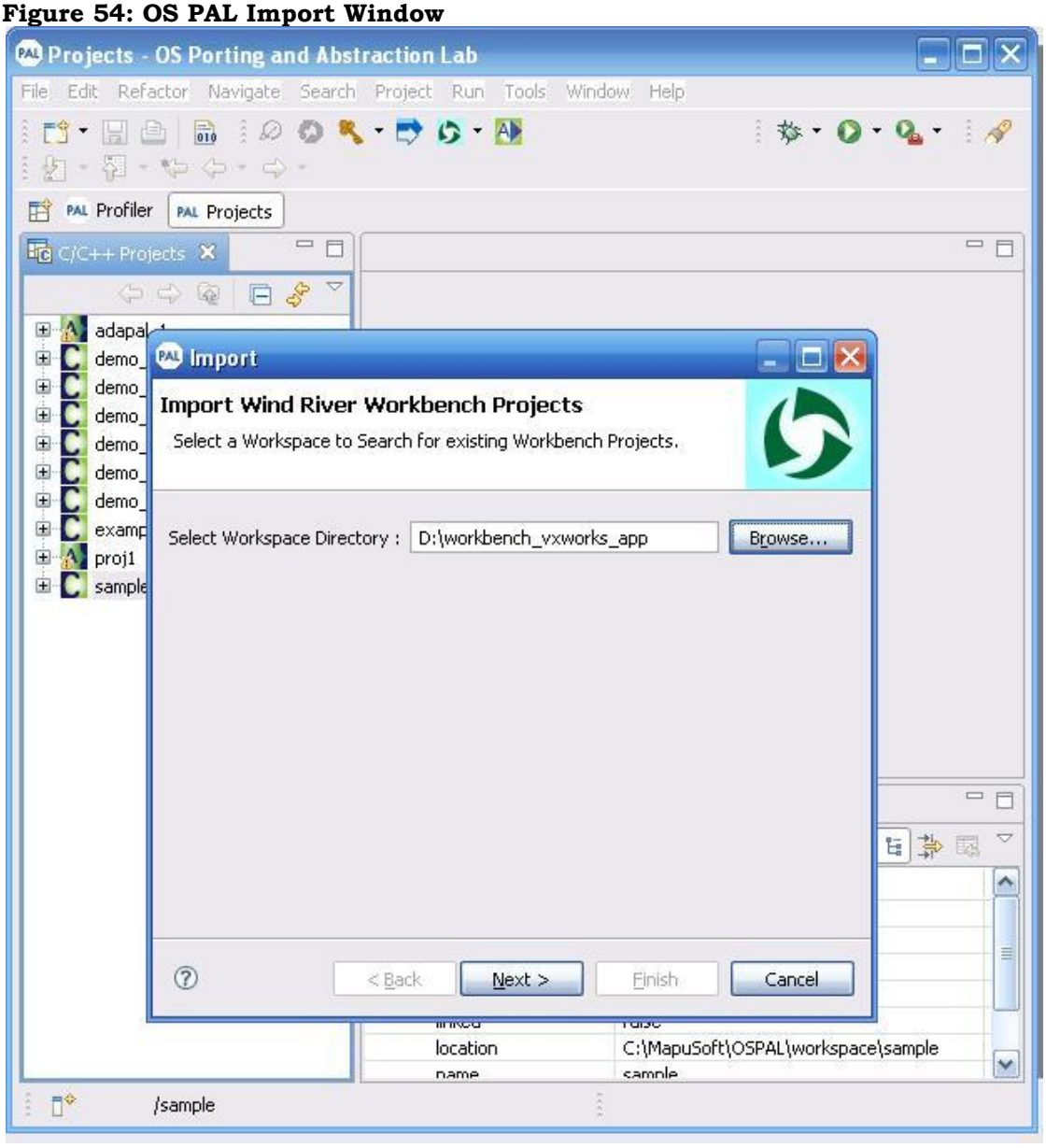
1. From OS PAL main window, select any project underC/C++ **Projects** tab on the left pane.
2. Select **File> Porting >VxWorks> Import Workbench 'C' Project** as shown in Figure 53. You can also click on the Porting icon  from the task bar.

**Figure 53: Importing a VxWorksWorkbench 'C' Project in OS PAL**



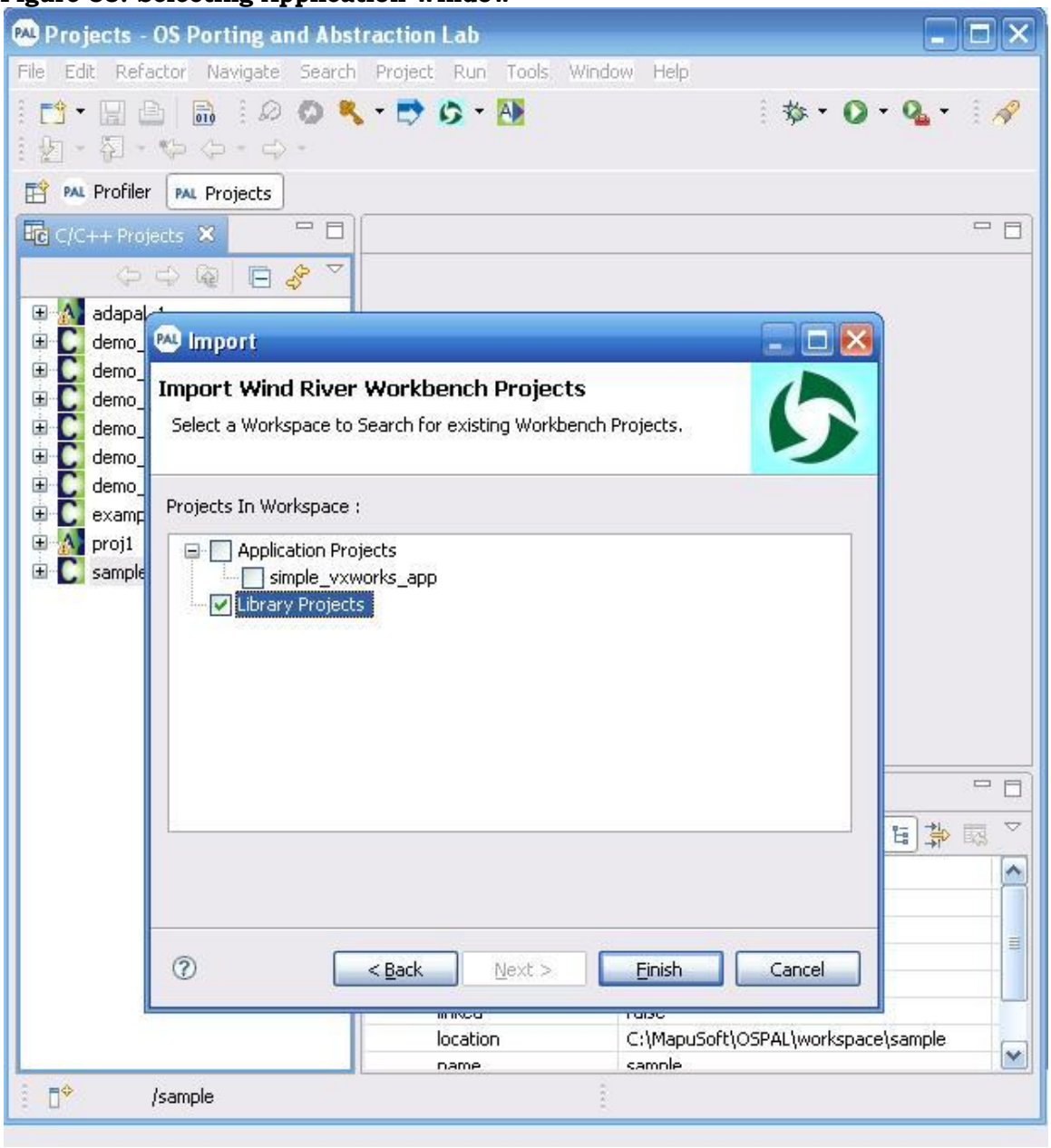
3. On OS PAL Import window, select a workspace directory to search for existing workbench projects by clicking on **Browse** button next to the text box, and click **Next** as shown inFigure 54.

**Figure 54: OS PAL Import Window**



4. In Import Wind River Workbench Projects window, the projects list is displayed in a checkbox Tree. Application projects and Library projects are separated into respective categories.
5. Select or deselect any one or all of the projects by selecting the check box next to the project name and click **Finish** to import the project as shown inFigure 55.

**Figure 55: Selecting Application Window**




The following are the project types:

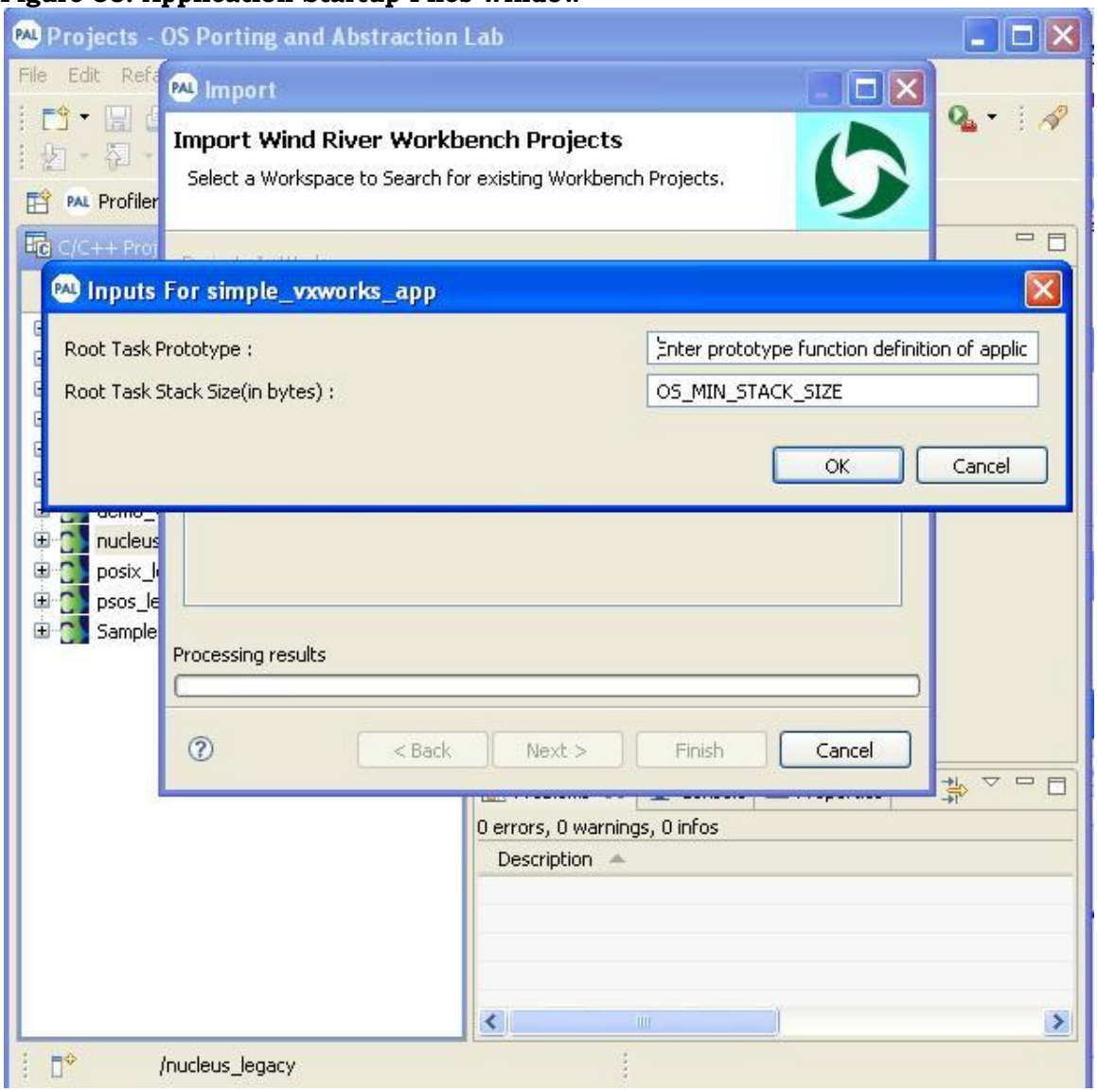
- **Application Projects** - Provides an executable application. This project type folder contains three templates. The makefile for the Executable project type is automatically created by the CDT.

- **Library Projects**- An executable module that is compiled and linked separately. When you create a project that uses a shared library (libxx.so), you define your shared library's project as a Project Reference for your application. The makefile for this project type is automatically created by the CDT.

**NOTE**: Select the check box next to your required library or application project to be imported.

6. If you select Application project, and click **Finish,** you get Application Start up Files windowas shown inFigure 56.

**Figure 56: Application Startup Files Window**

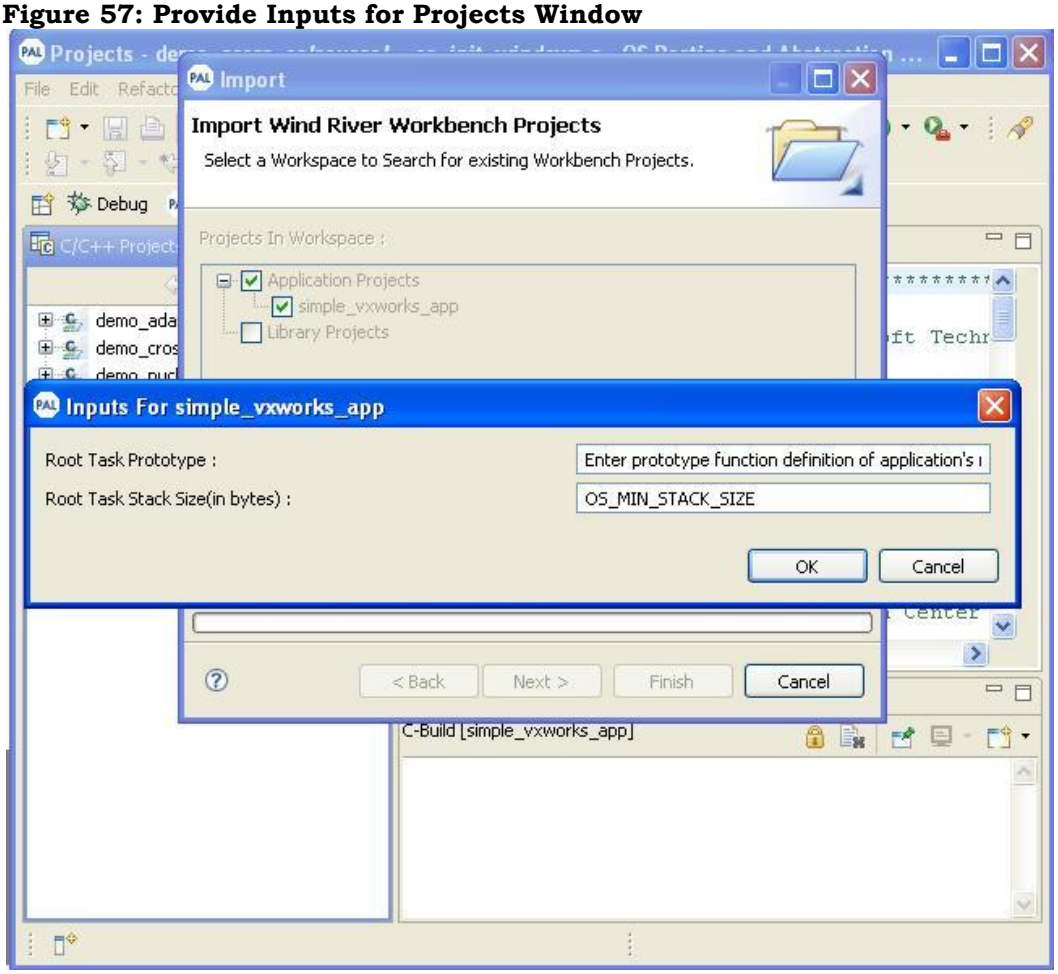



7. If you are importing a kernel application, click **Yes** to automatically create start up files to connect the imported application to the OS platformas shown in the Figure 56.

**NOTE:** If you are porting a library project, click **No** to continue with the porting.

8. If you select any application type project, provide the inputs for the project and click **OK** as shown inFigure 57. If you do not want to provide the inputs, you can just click **Cancel**.

**Figure 57: Provide Inputs for Projects Window**



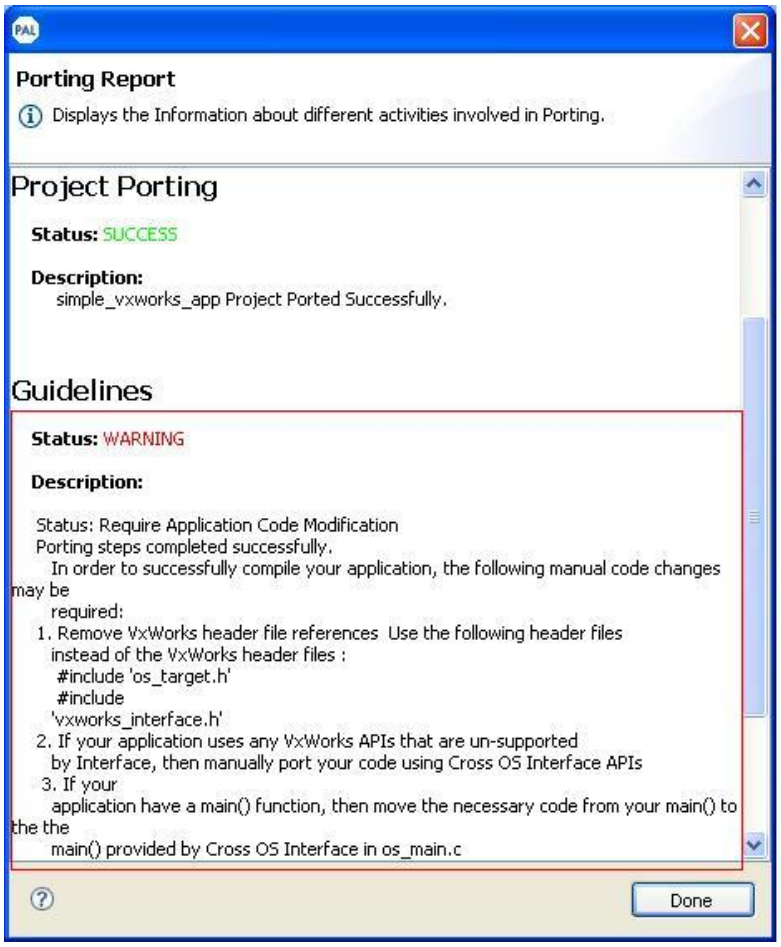
**NOTE**: If you select an application project and if it contains any referenced projects not selected by you, then a Confirmation dialogue box is displayed on your screen to ask if you want to port the project. If you want to port, click **OK**. You can see the porting processing results on the window.

9. After the porting is successfully done, the porting report page is displayed as shown inFigure 58. Click **Done** to complete the process.

**Figure 58: Porting Reports Page**

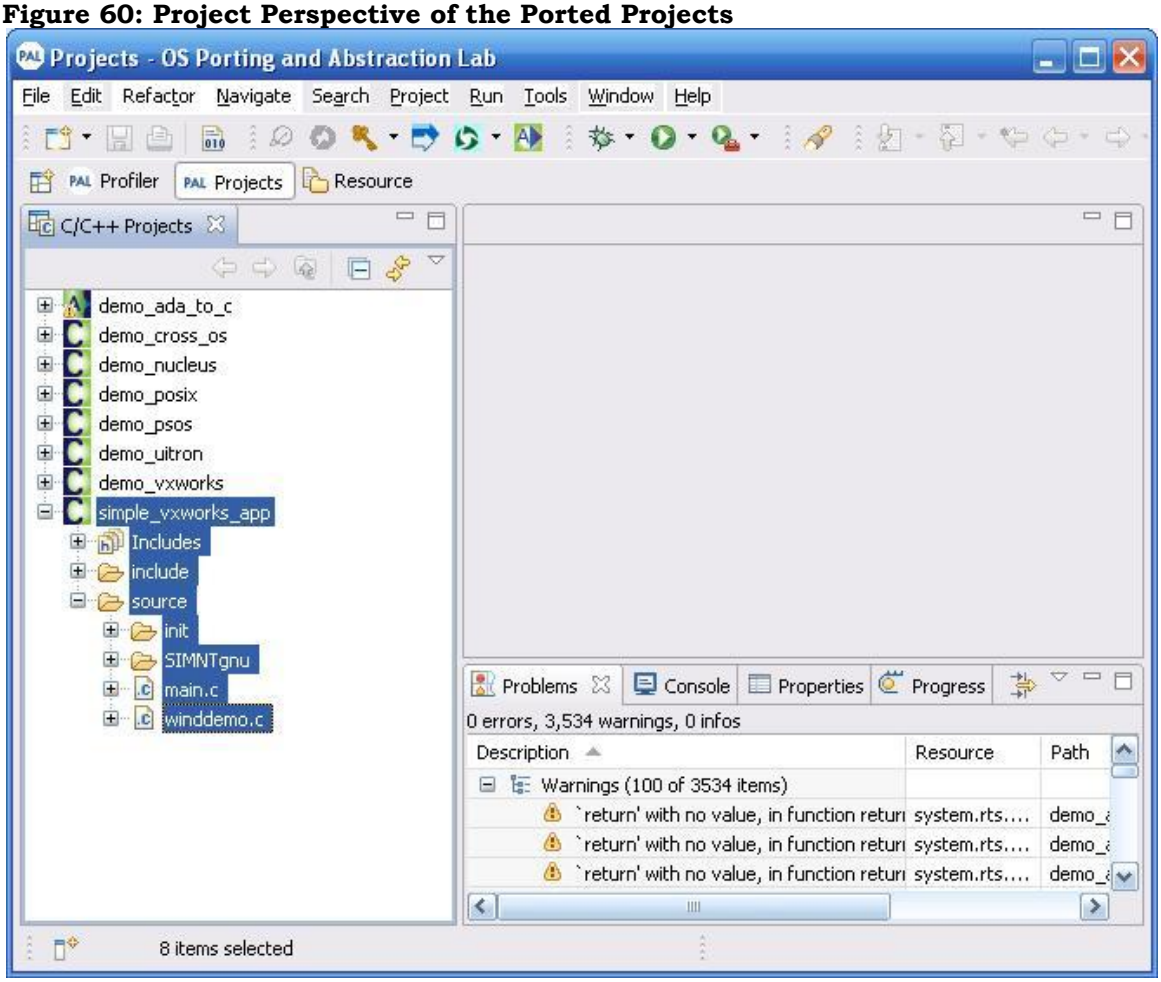
10. In order to successfully compile your application, follow the guidelines highlighted inFigure 59.

**Figure 59: Porting Reports Page_ Guidelines**



11. In OS PAL projects perspective, the ported projects are displayed as shown in Figure 60.

**Figure 60: Project Perspective of the Ported Projects**




You have successfully imported your VxWorks application to OS PAL.

To know more about the project template files, go to OS PAL Project Template Files on page 60.

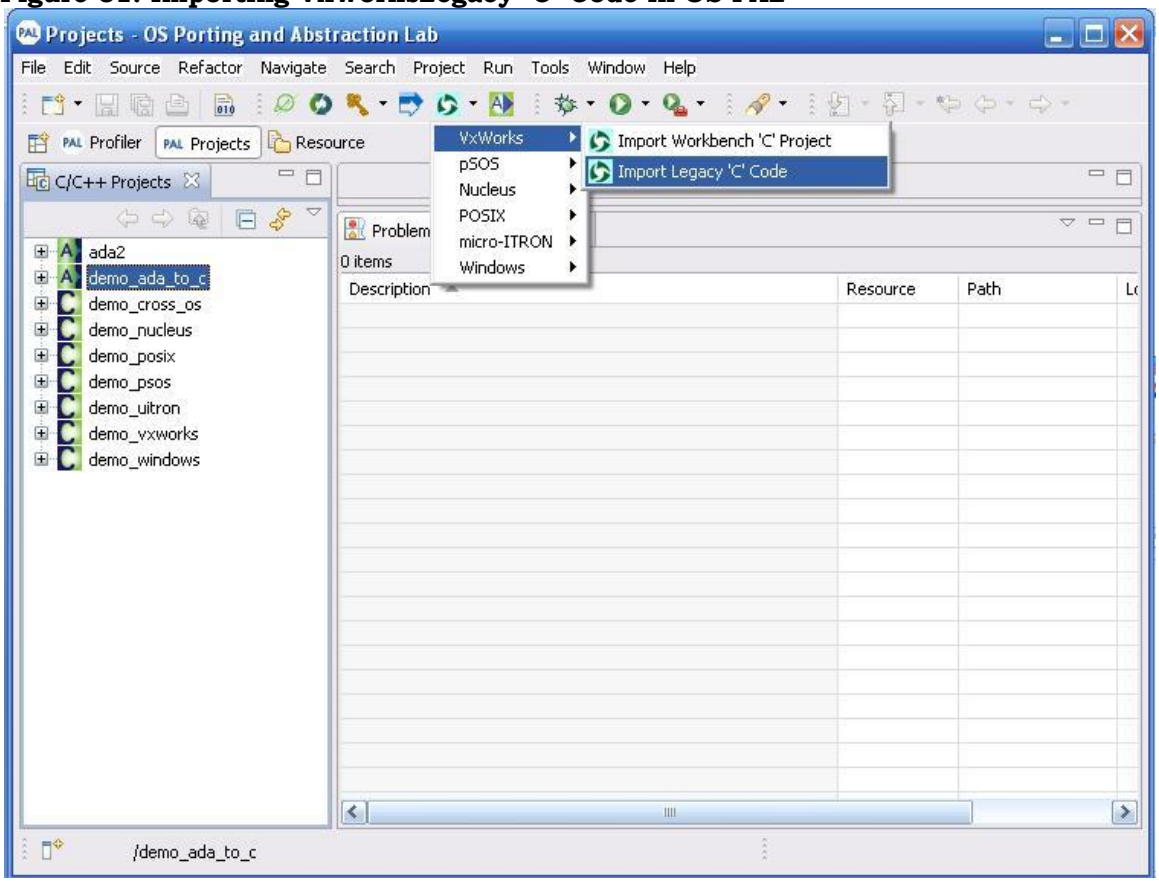
## Method 2–Porting VxWorksLegacy 'C' Code

This section explains Porting VxWorksLegacy Applications using OS PAL Porting Plugin. A sample porting of VxWorks Legacy application using OSPAL is described with an example here.

**NOTE**: This feature requires a license. Click http://mapusoft.com/downloads/ospal-evaluation/ to request an evaluation license.

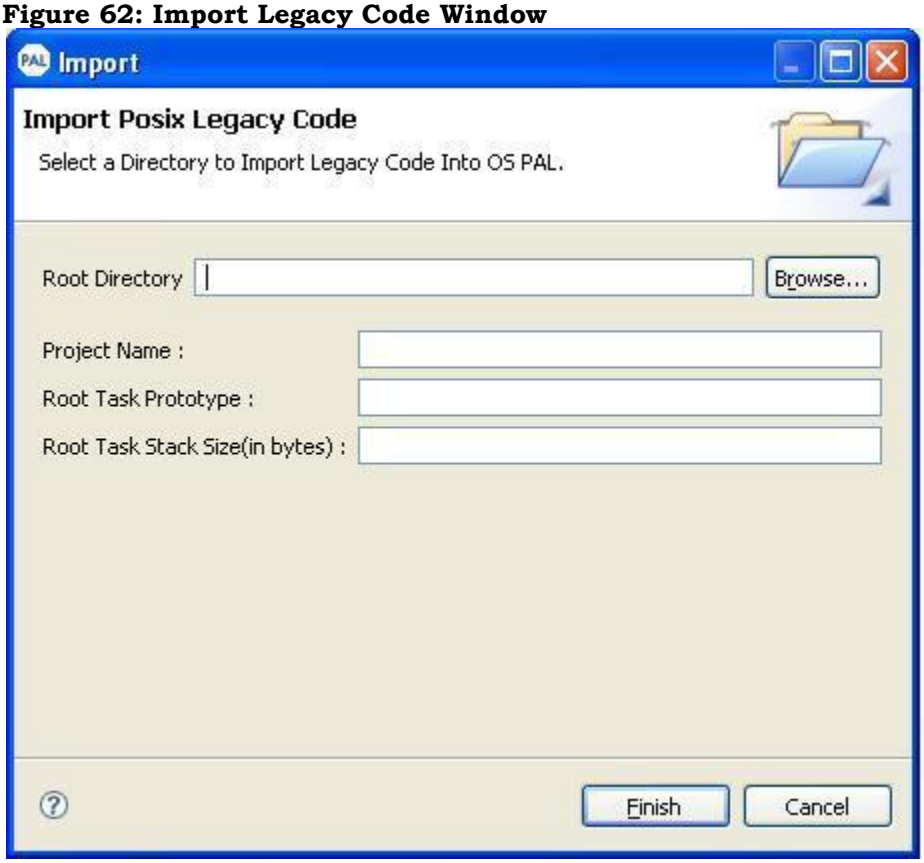
1. Select **File> Porting >VxWorks> Import Legacy 'C' Code** as shown inFigure 61. You can also click on the Porting icon  from the task bar.

**Figure 61: Importing VxWorksLegacy 'C' Code in OS PAL**



2. On OS PAL Import window, select the root directory from where you want to import the legacy code by clicking on **Browse** button next to the text box, and click **Next** as shown inFigure 62.

**Figure 62: Import Legacy Code Window**




3. Enter the project name for which you want to import the legacy code in the **Project Name** text box.
4. Enter the root task prototype, next to **Root Task Prototype** text box.
5. Enter the root task stack size, next to the **Root Task Stack Size** text box. The value should be in bytes.
6. Click **Finish** to complete the importing of legacy code into OS PAL.

You have successfully imported the legacy code and a project with your given project name is created in the current workspace.

## Method 3– Manually Porting Legacy Applications using OS PAL Import Feature

**Step 1:** In your source code, remove references to the original OS include files and use os_target.h and the header files of your ported legacy API instead.

1. Remove the original OS specific initialization code and use OS_Application_Init function call instead (refer to the Cross-OS Interface Reference Manual).
2. Create an OS PAL project for your legacy application and select the legacy API that your application will need (Eg: to port VxWorks application, you need to check the "Include VxWorks Interface API's").
3. If your application uses any APIs that are not supported under OS PAL, re-write the code using Cross-OS Interface APIs.
4. Import your legacy application into the new project.
5. Compile and link your application and resolve all compiler and linker errors.
6. Run or debug your application under OS PAL host in an x86 environment. You should rewrite/replace any hardware specific code in your application for this step.

**Step 2**: Moving from OS PAL Host to target using OS PAL Optimized Target Code Generator:

1) Generate the code for your target OS using the OS PAL Optimized Target Code Generator.
2) Using cross-compiler compile, link, and download the OS PAL generated code to your target.
3) Port low level drivers and hardware interrupt code as required (refer to Cross-OS Interface I/O and device driver APIs sections in the reference manual).
4) Resolve any run time errors.

# Porting POSIX Legacy 'C' Code

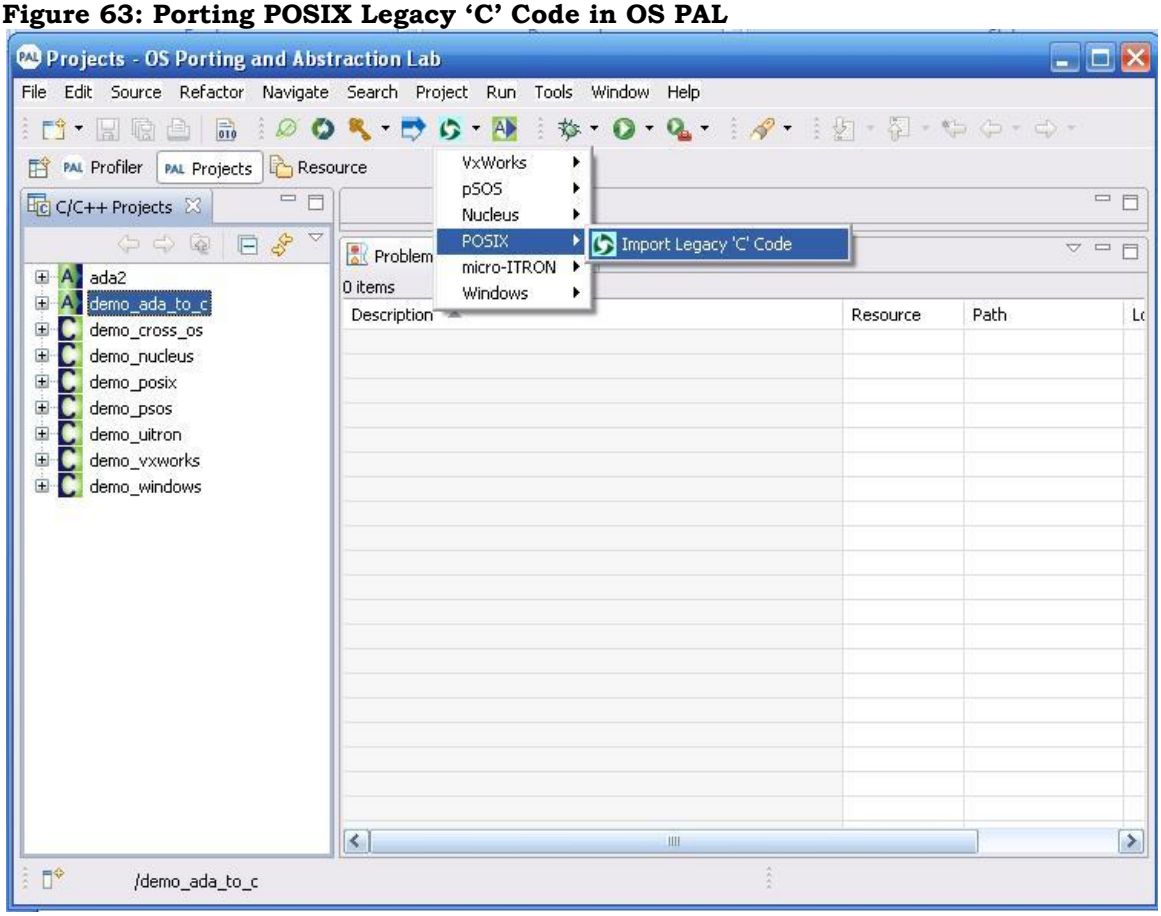
This section explains Porting POSIX Legacy Applications using OS PAL Porting Plugin. A sample porting of POSIX Legacy applications using OSPAL is described with an example here.

**NOTE**: This feature requires a license. Click http://mapusoft.com/downloads/ospal-evaluation/ to request an evaluation license.

To port a sample POSIX legacy application:

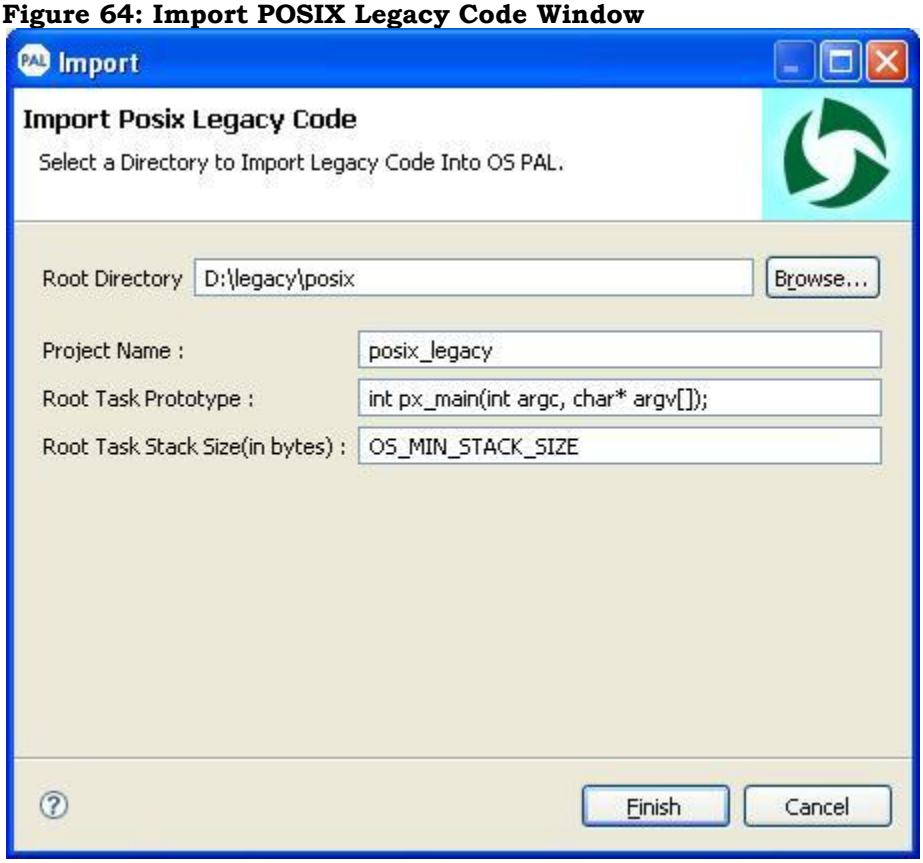
1. Select File**> Porting >POSIX> Import Legacy 'C' Code** as shown inFigure 63. You can also click on the Porting icon from the task bar.

**Figure 63: Porting POSIX Legacy 'C' Code in OS PAL**



2. On OS PAL Import window, select the root directory from where you want to import the legacy code by clicking on **Browse** button next to the text box, and click **Next** as shown in Figure 64.

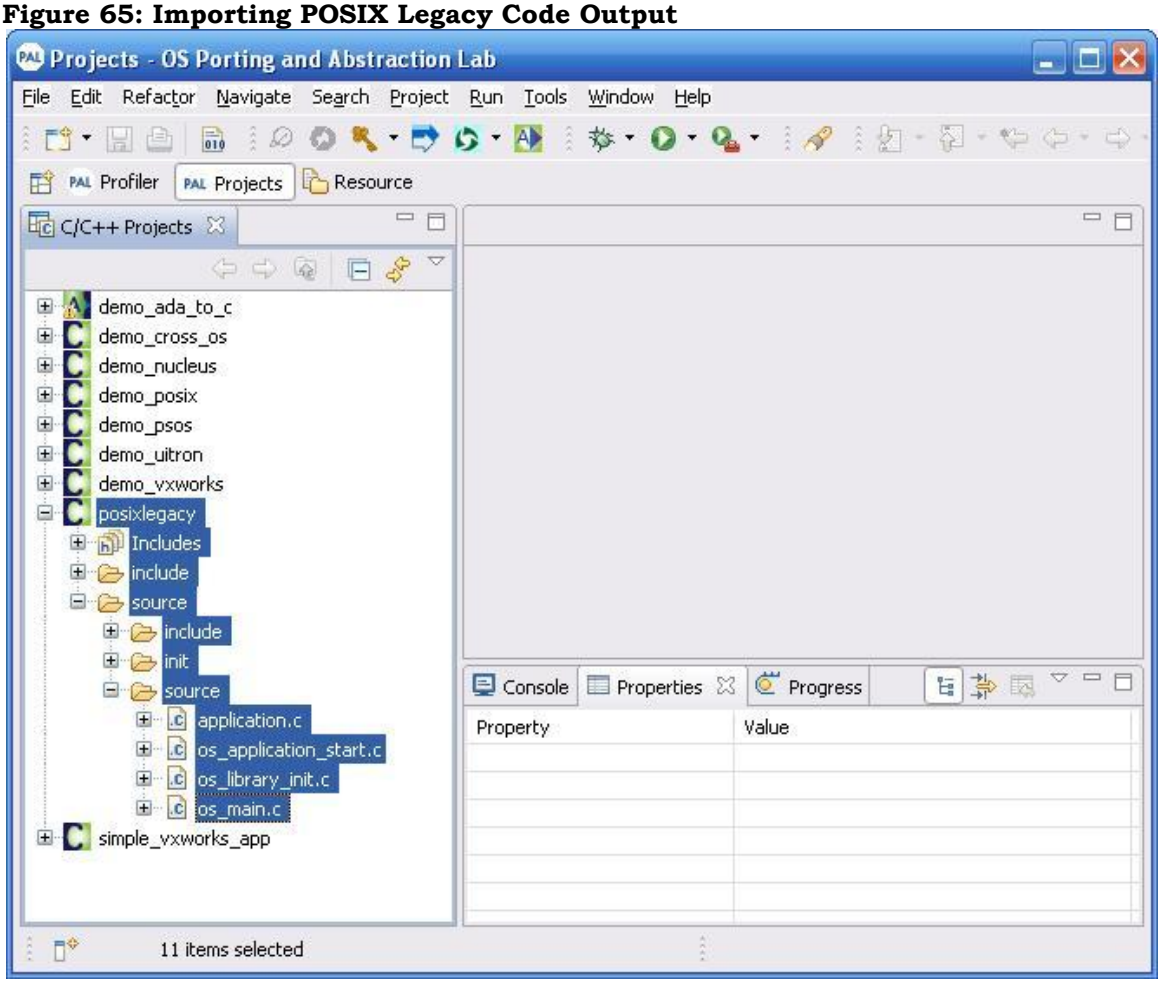
**Figure 64: Import POSIX Legacy Code Window**




3. Enter the project name for which you want to import the legacy code in the **Project Name** text box.
4. Enter the root task prototype, next to **Root Task Prototype** text box.
5. Enter the root task stack size, next to the **Root Task Stack Size** text box. The value should be in bytes.

6. Click **Finish** to complete the importing of legacy code into OS PAL. You can see POSIX legacy code you have imported as shown in Figure 65.

**Figure 65: Importing POSIX Legacy Code Output**

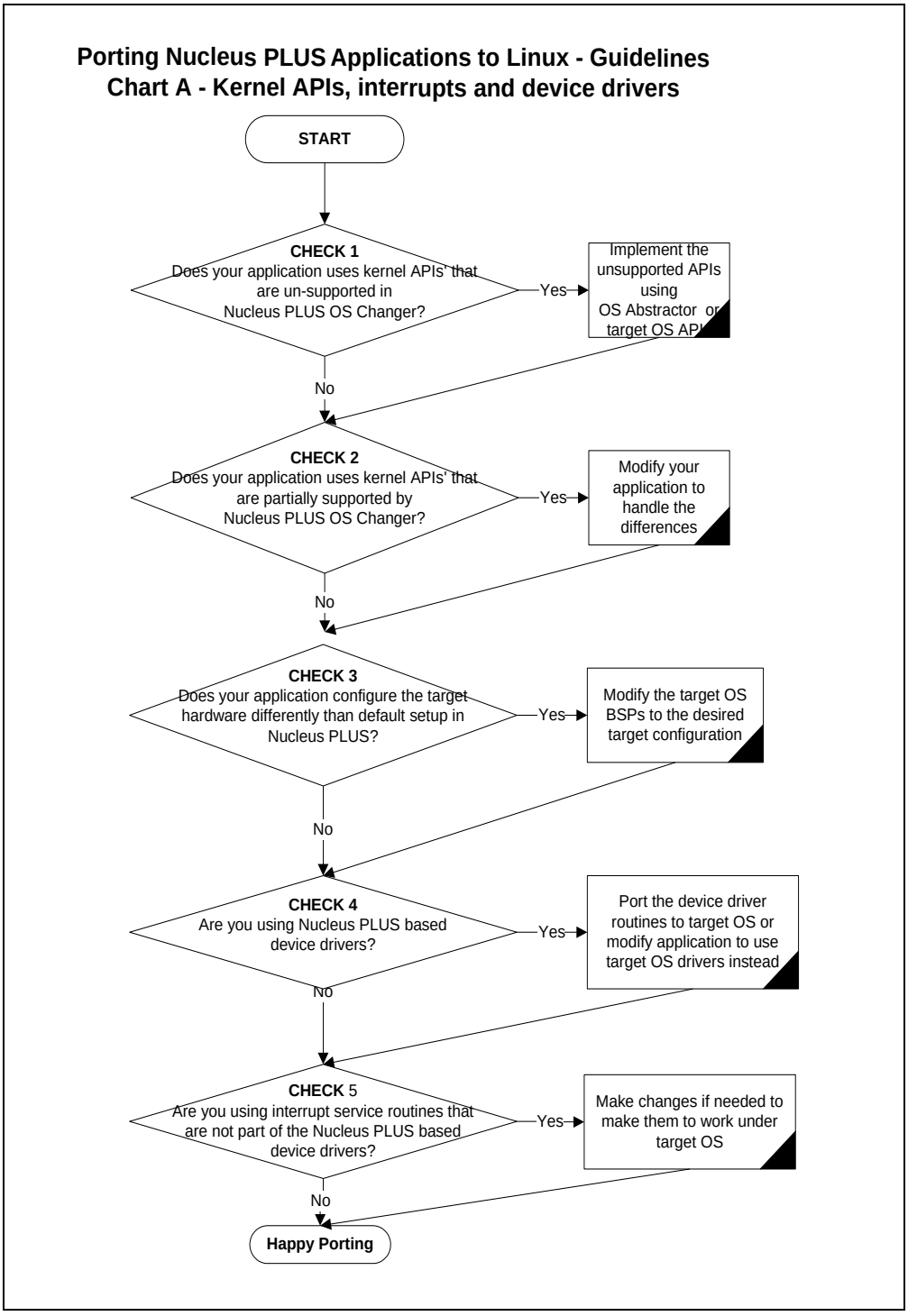



You have successfully imported the POSIX legacy C code and a project with your given project name is created in the current workspace.

# Porting Applications from Nucleus PLUS Legacy Code to Target OS

In most applications, using Nucleus OS Changer is straightforward. The effort required in porting is mostly at the underlying driver layer. Since we do not have specific information about your application, it will be hard to tell how much work is required. However, we want you to be fully aware of the surrounding issues upfront so that necessary steps could be taken for a successful and timely porting. This section provides porting guidelines in a flow chart format. This covers issues relating with Nucleus OS Changer, device drivers, interrupt service routines, etc. It is possible that we have not addressed all your application specific issues in the flow chart, so for further information, contact MapuSoft Technologies.

**Figure 66: Porting Nucleus PLUS Applications**



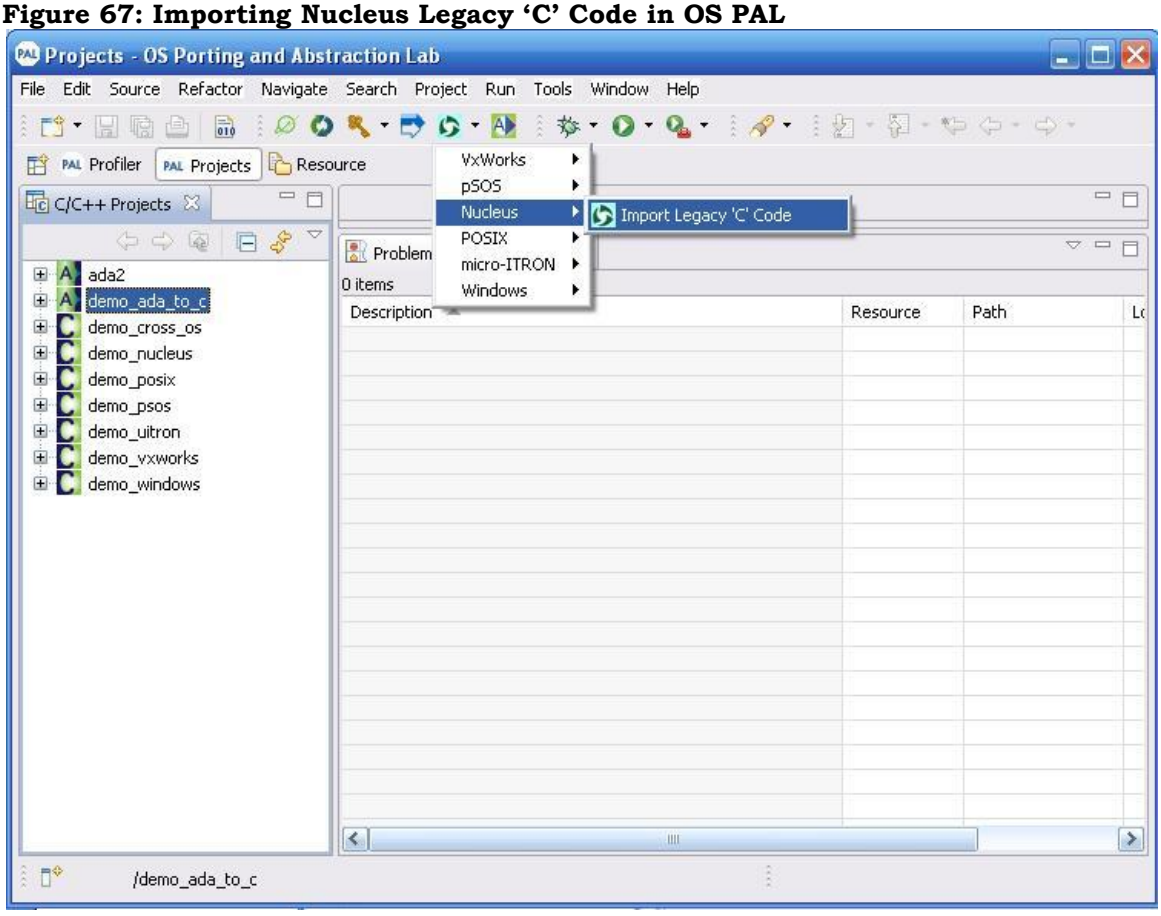
# Porting Nucleus Legacy 'C' Code

This section explains Porting Nucleus Legacy Applications using OS PAL Porting Plugin. A sample porting of Nucleus Legacy application using OSPAL is described with an example here.

**NOTE**: This feature requires a license. Click http://mapusoft.com/downloads/ospal-evaluation/ to request an evaluation license.

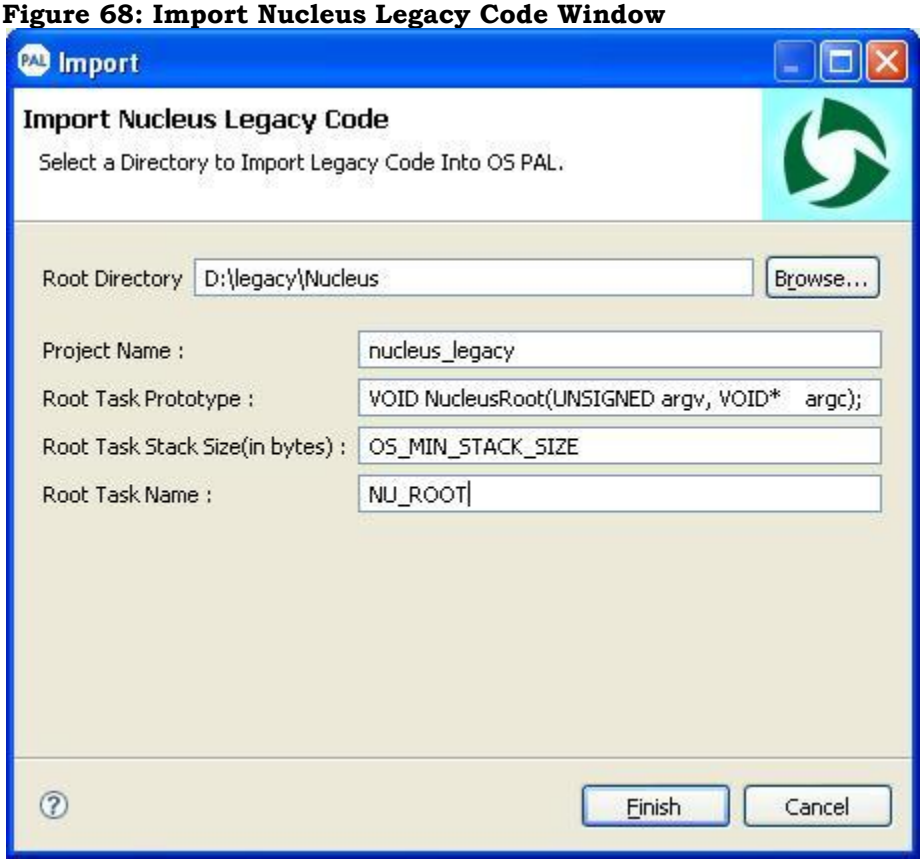
1. Select File**> Porting > Nucleus > Import Legacy 'C' Code** as shown inFigure 67. You can also click on the Porting icon from the task bar.

**Figure 67: Importing Nucleus Legacy 'C' Code in OS PAL**



2. On OS PAL Import window, select the root directory from where you want to import the legacy code by clicking on **Browse** button next to the text box, and click **Next** as shown inFigure 68.

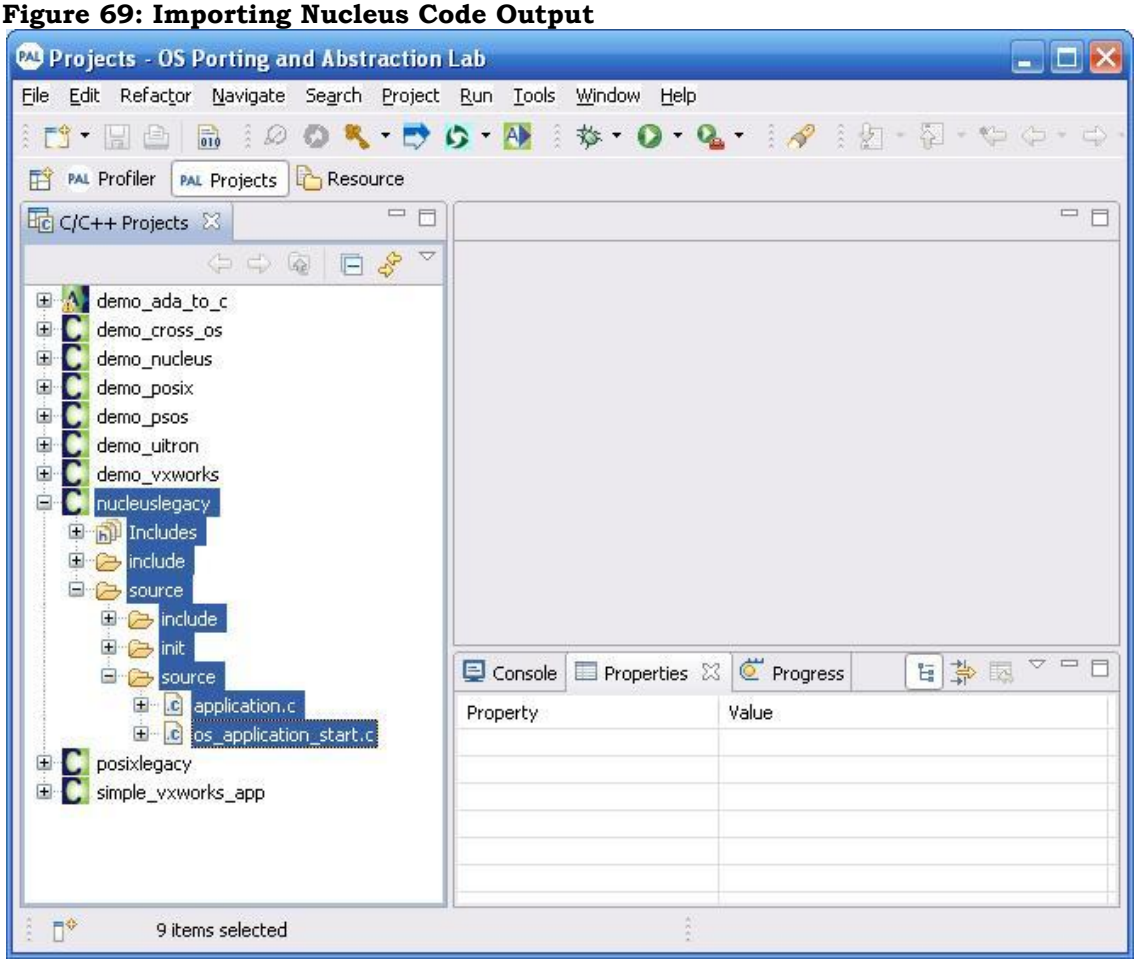
**Figure 68: Import Nucleus Legacy Code Window**




3. Enter the project name for which you want to import the legacy code in the **Project Name** text box.
4. Enter the root task prototype, next to **Root Task Prototype** text box.
5. Enter the root task stack size, next to the **Root Task Stack Size** text box. The value should be in bytes.
6. Enter the root task name, next to **Root Task Name**text box.

7. Click **Finish** to complete the importing of legacy code into OS PAL. You can see the output as shown in Figure 69.

**Figure 69: Importing Nucleus Code Output**




You have successfully imported Nucleus legacy C code and a project with your given project name is created in the current workspace.

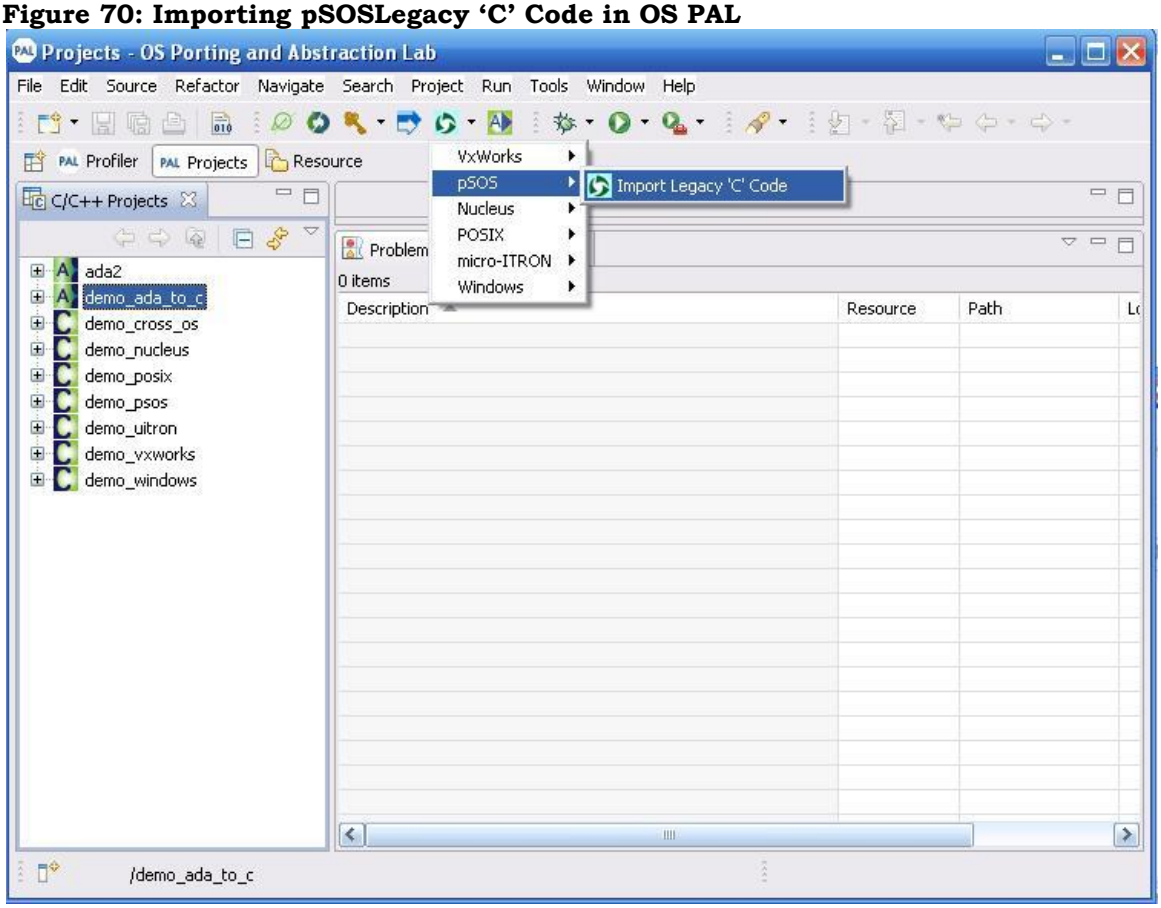
# Porting pSOS Legacy 'C' Code

This section describes the sample porting of pSOS Legacy Applications using OS PAL Porting Plugin. A description for porting pSOS Legacy applications using OSPAL is described with an example here.

**NOTE**: This feature requires a license. Click http://mapusoft.com/downloads/ospal-evaluation/ to request an evaluation license.

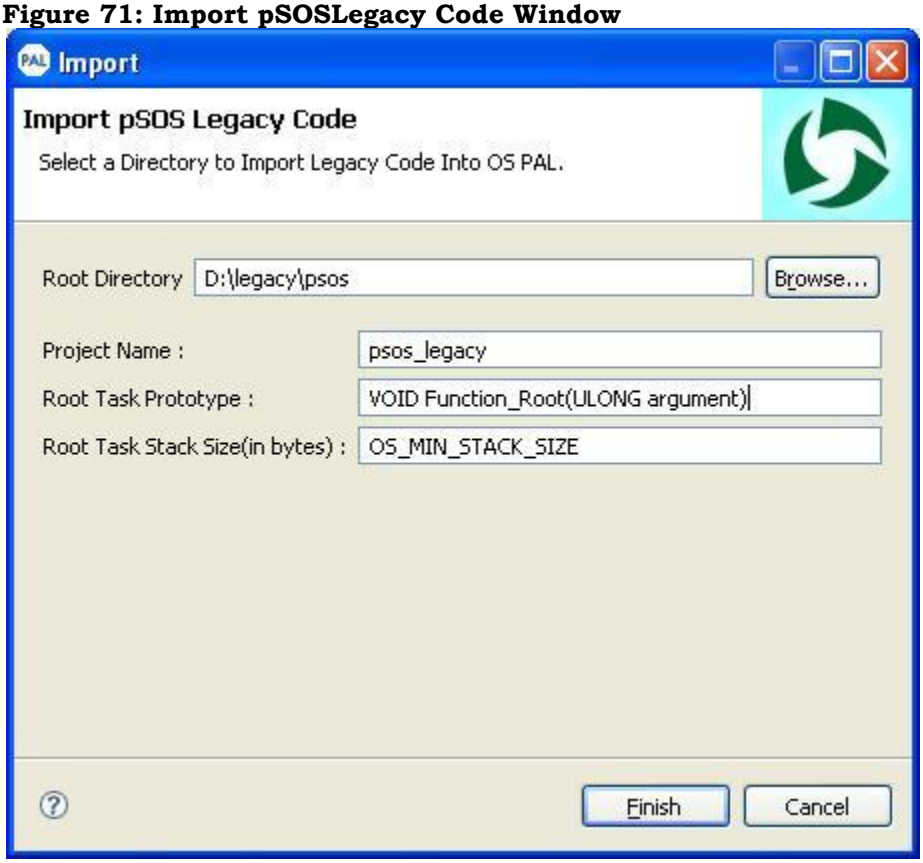
1. Select **File> Porting >pSOS> Import Legacy 'C' Code** as shown inFigure 70. You can also click on the Porting icon from the task bar.

**Figure 70: Importing pSOSLegacy 'C' Code in OS PAL**



2. On OS PAL Import window, select the root directory from where you want to import the legacy code by clicking on **Browse** button next to the text box, and click **Next** as shown inFigure 71.

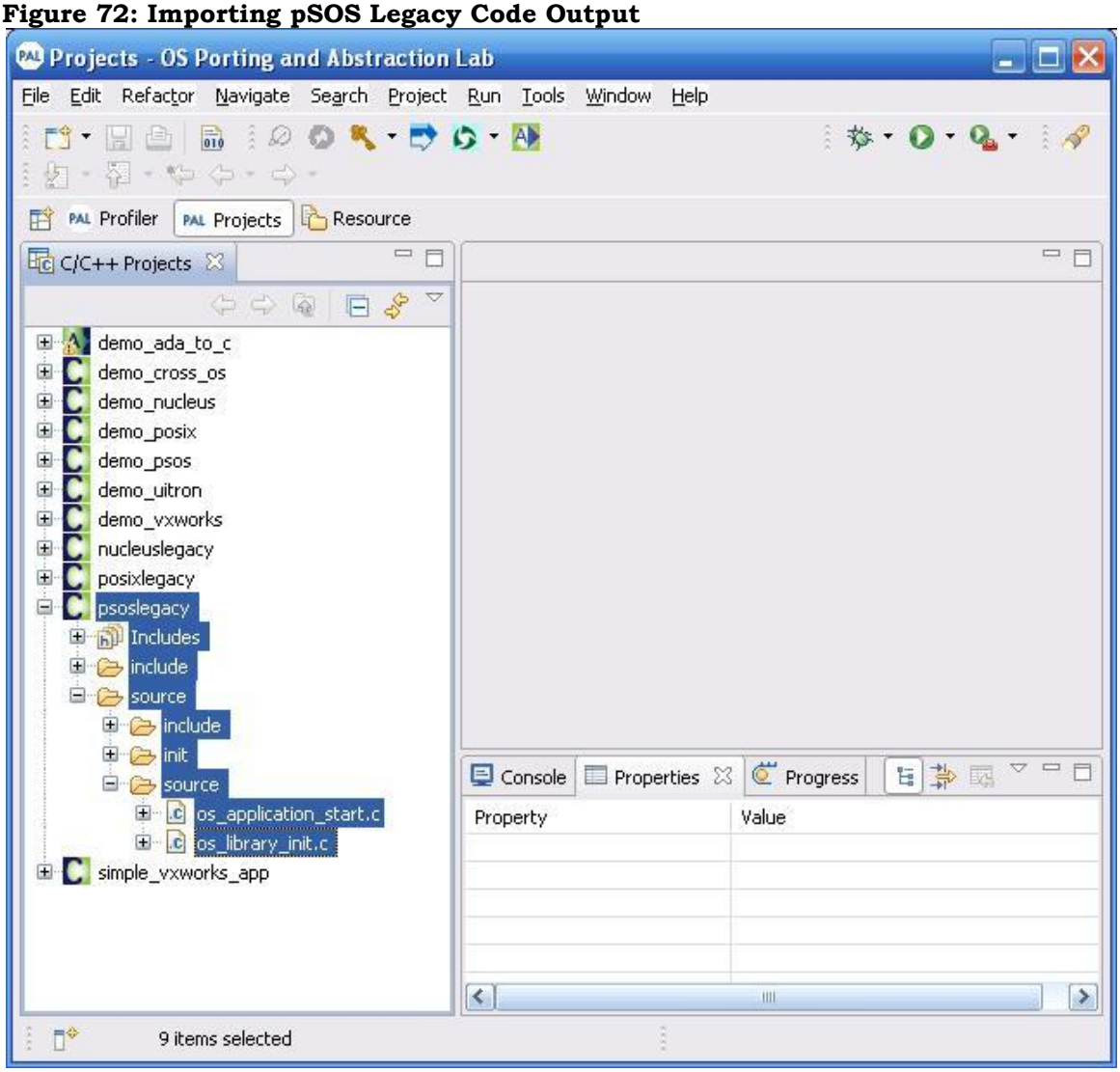
**Figure 71: Import pSOSLegacy Code Window**




3. Enter the project name for which you want to import the legacy code in the **Project Name** text box as shown in the figure.
4. Enter the root task prototype, next to **Root Task Prototype** text box as shown in the figure.
5. Enter the root task stack size, next to the **Root Task Stack Size** text box as shown in the figure. The value should be in bytes.

6. Click **Finish** to complete the importing of legacy code into OS PAL. You can see POSIX legacy code you have imported as shown in Figure 72.

**Figure 72: Importing pSOS Legacy Code Output**




You have successfully imported pSOSlegacy C code and a project with your given project name is created in the current workspace.

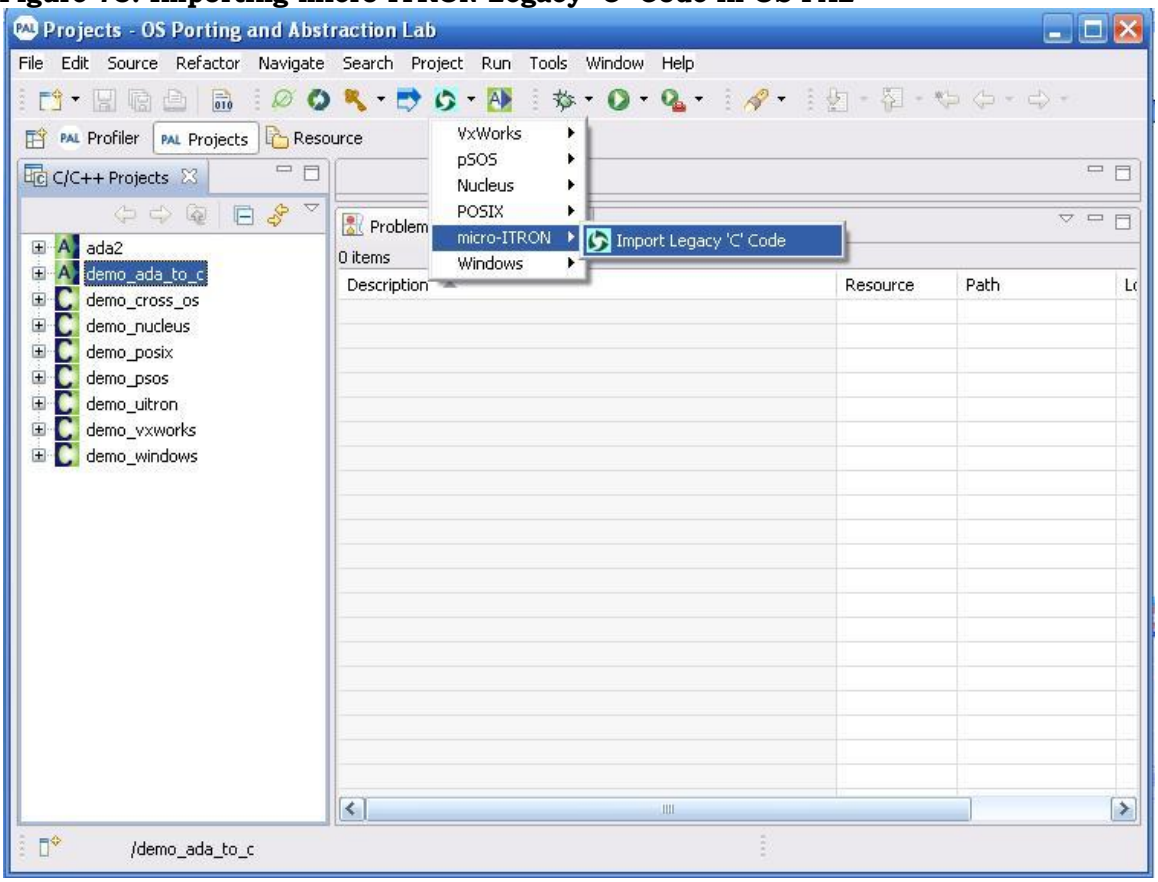
# Porting micro-ITRON Legacy 'C' Code

This section explains porting of micro-ITRON Legacy Applications using OS PAL Porting Plugin. A sample porting of micro-ITRON Legacy applications using OSPAL is described with an example here.

**NOTE**: This feature requires a license. Click http://mapusoft.com/downloads/ospal-evaluation/ to request an evaluation license.

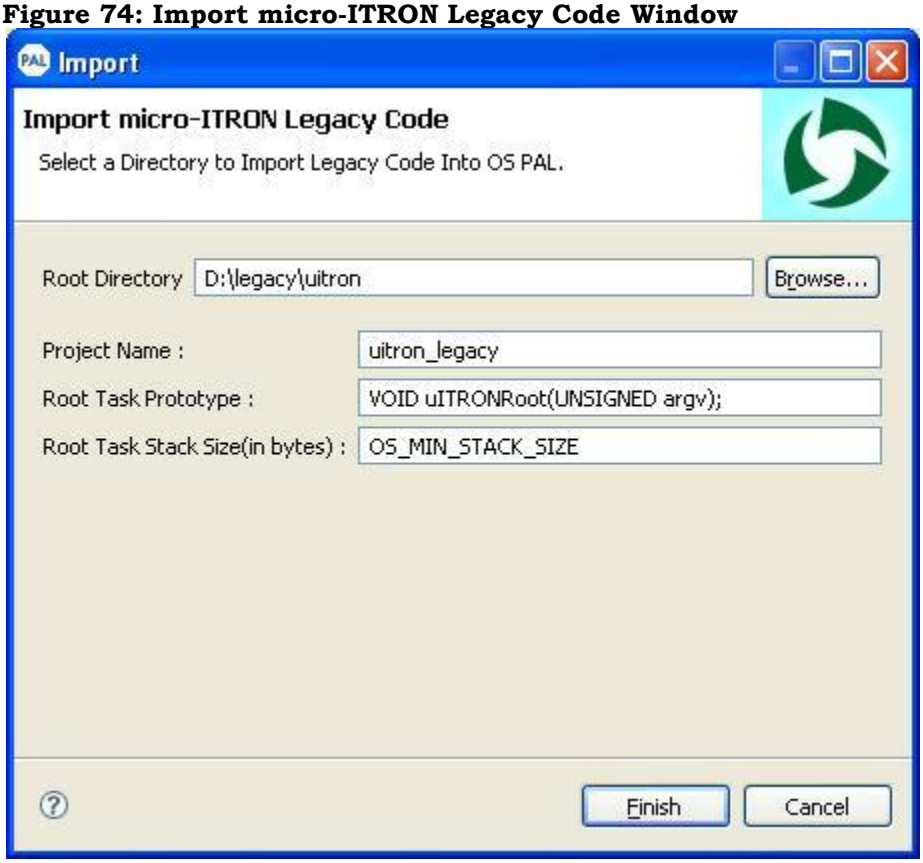
1. Select **File> Porting >micro-ITRON> Import Legacy 'C' Code** as shown inFigure 73. You can also click on the Porting icon  from the task bar.

**Figure 73: Importing micro-ITRON Legacy 'C' Code in OS PAL**



2. On OS PAL Import window, select the root directory from where you want to import the legacy code by clicking on **Browse** button next to the text box, and click **Next** as shown inFigure 74.

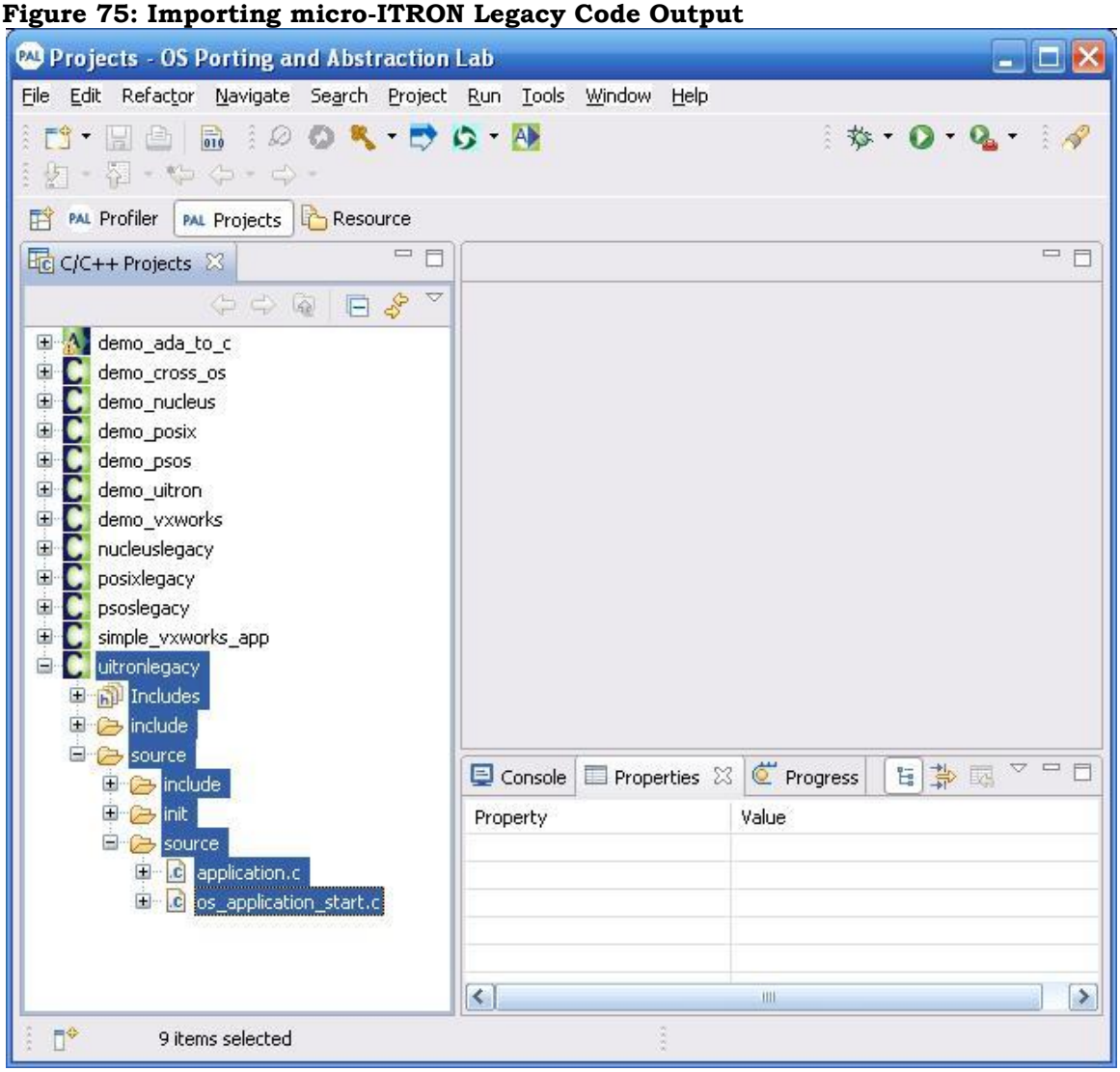
**Figure 74: Import micro-ITRON Legacy Code Window**




3. Enter the project name for which you want to import the legacy code in the **Project Name** text box as shown in the figure.
4. Enter the root task prototype, next to **Root Task Prototype** text box as shown in the figure.
5. Enter the root task stack size, next to the **Root Task Stack Size** text box as shown in the figure. The value should be in bytes.

6. Click **Finish** to complete the importing of legacy code into OS PAL. You can see micro-ITRON legacy code you have imported as shown in Figure 75.

**Figure 75: Importing micro-ITRON Legacy Code Output**




You have successfully imported micro-ITRON legacy C code and a project with your given project name is created in the current workspace.

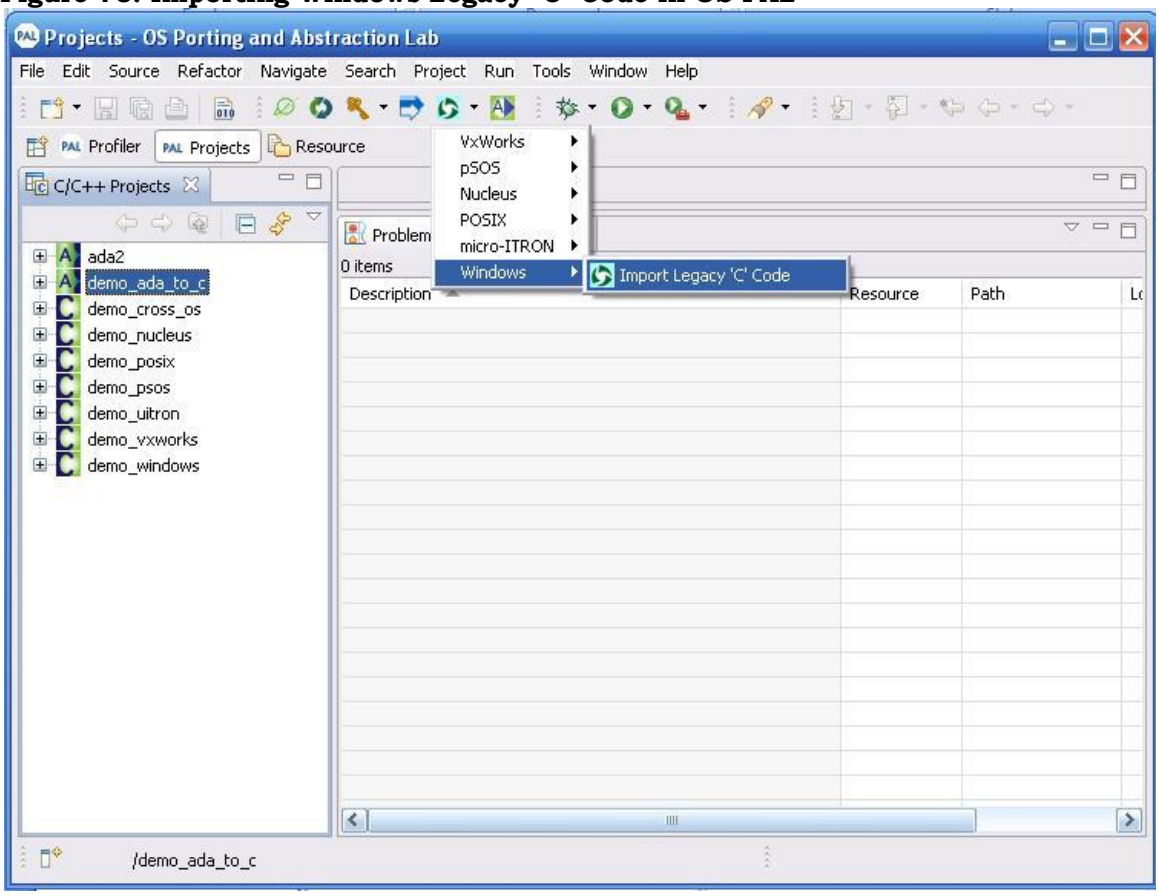
# Porting Windows Legacy 'C' Code

This section explains porting of Windows Legacy Applications using OS PAL Porting Plugin. A sample porting of Windows Legacy applications using OSPAL is described with an example here.

**NOTE**: This feature requires a license. Click http://mapusoft.com/downloads/ospal-evaluation/ to request an evaluation license.

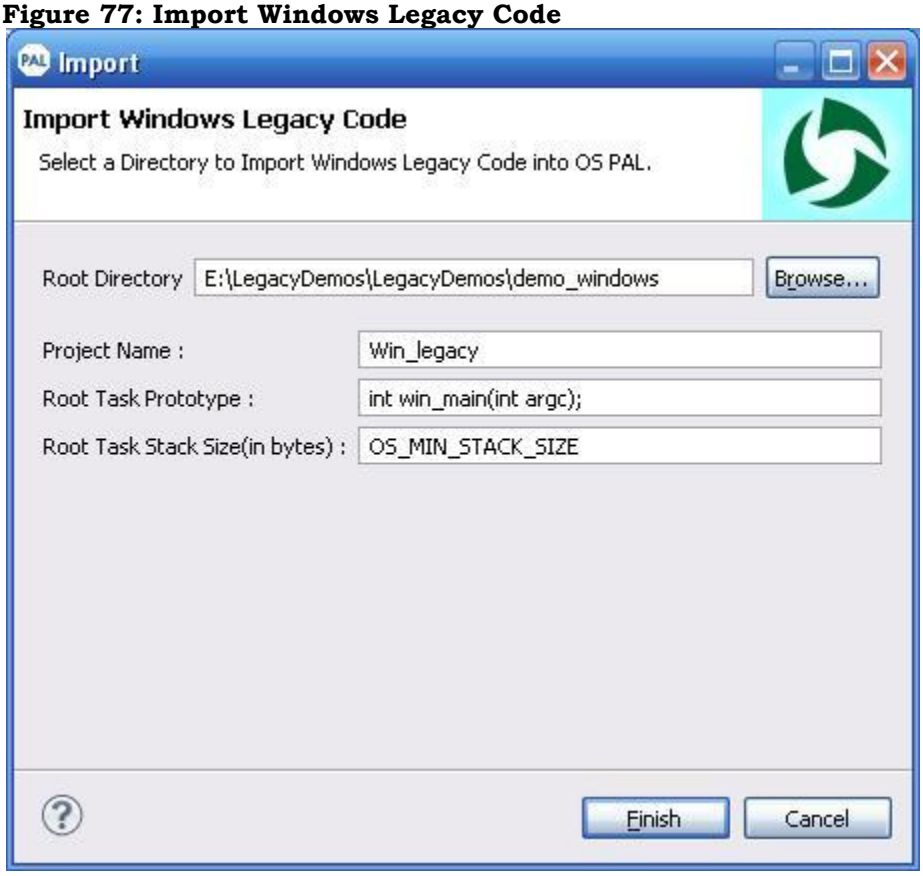
1. Select **File> Porting >Windows> Import Legacy 'C' Code** as shown in Figure 76. You can also click on the Porting icon  from the task bar.

**Figure 76: Importing Windows Legacy 'C' Code in OS PAL**



2. On OS PAL Import window, select the root directory from where you want to import the legacy code by clicking on **Browse** button next to the text box, and click **Next** as shown in Figure 77.

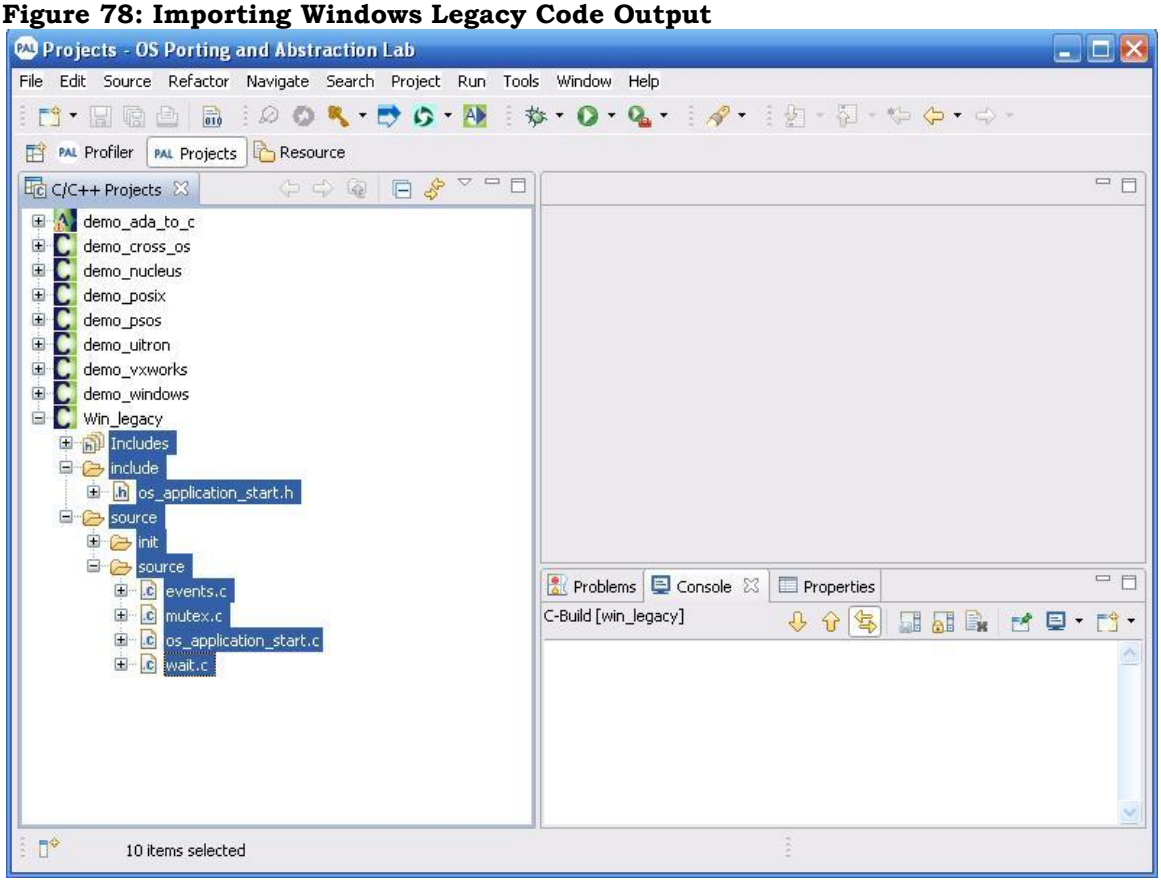
**Figure 77: Import Windows Legacy Code**




3. Enter the project name for which you want to import the legacy code in the **Project Name** text box as shown in the figure.
4. Enter the root task prototype like *intwin_main(intargc);* next to **Root Task Prototype** text box as shown in the figure.
5. Enter the root task stack size, next to the **Root Task Stack Size** text box as shown in the figure. The value should be in bytes.

6. Click **Finish** to complete the importing of legacy code into OS PAL. You can see Windows legacy C code you have imported as shown in Figure 78.

**Figure 78: Importing Windows Legacy Code Output**




You have successfully imported Windows legacy code and a project with your given project name is created in the current workspace.

# Chapter 4. OS Changer for Reusing the Code

OS Changer allows you to reuse the code on any new OS without having to rewrite or port your code. This saves time and money by reducing the porting effort.

OS Changer provides extensive support to various common proprietary libraries widely used by the application developers. Further, developers can use the native TARGET OS interface as well. This works toward getting the migration effort faster and much easier.

This chapter contains the following topics:

# About OS Changer

OS Changer is designed for use as a C library. Services used inside your application software are extracted from the OS Changer and TARGET OS libraries, and, are then combined with the other application objects to produce the complete image.

OS Changer is optimized to take full advantage of the underlying TARGET RTOS features. It is built to be totally independent of the target hardware and all the development tools (like compilers and debuggers).

Please note that there may be some minor implementation differences in some of the OS Changer APIs when compared to the native API's. This may be as a result of any missing features within the underlying RTOS that OS Changer provides migration to.

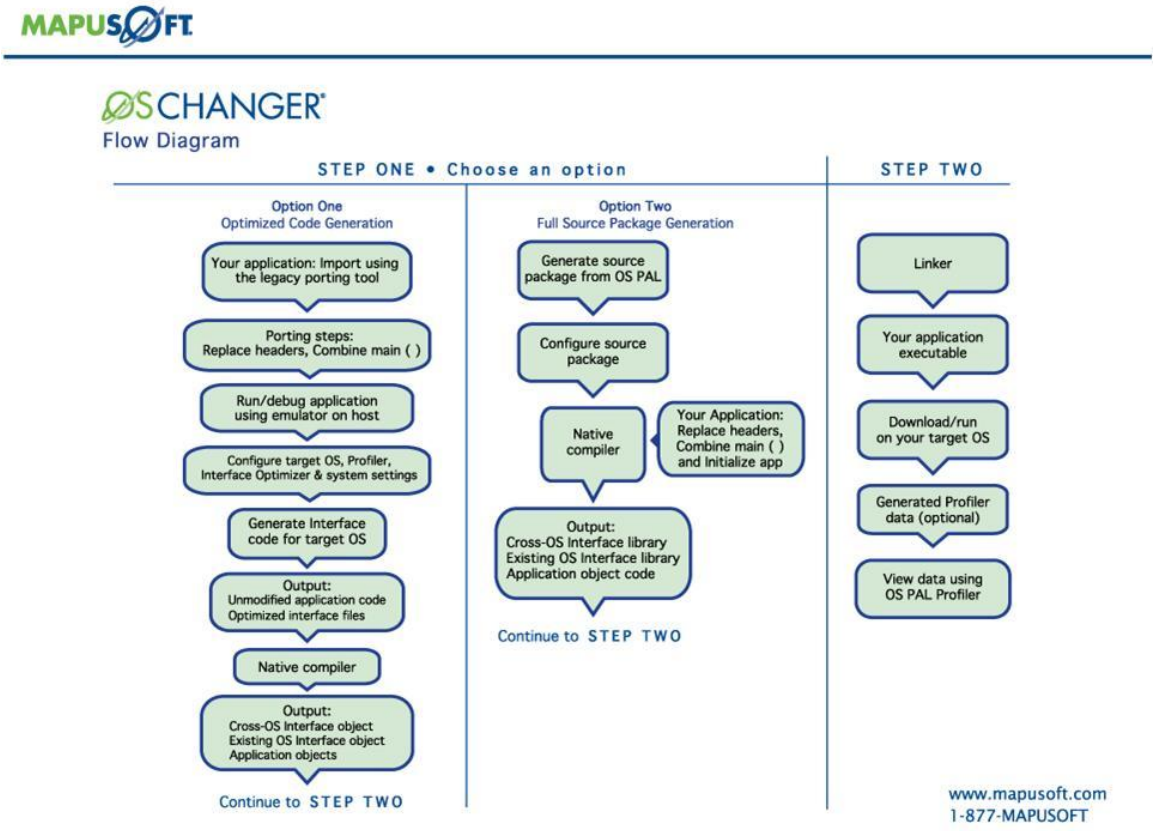



**Figure 79: *OS Changer Flow Diagram***

Your legacy application can be re-usable and also portable by the support provided by the OS Changer library and the OS Abstractor library. Applications can directly use the native target OS API, however doing so will not make your code portable across operating systems. We recommend that you use the optimized abstraction APIs for the features and support that are not provided by the OS Changer compatibility library.

**NOTE**: For more information on configuration and target OS specific information, see Cross-OS Interface Reference Manual.

# Interfaces Available for OS Changer

The following are the interfaces available for OS Changer:

- VxWorks
- Nucleus
- pSOS
- micro-ITRON
- POSIX
- Windows

# Using OS Changer

OS Changer is designed for use as a C library. Services used inside your application software are extracted from the OS Changer and TARGET OS libraries, and, are then combined with the other application objects to produce the complete image. This image can be loaded to the target system or placed in ROM on the target system.

The steps for using OS Changer are described in the following generic form:

- Remove the TARGET RTOS header file defines from all the TARGET RTOS source files.
- Remove definitions and references to all the TARGET RTOS configuration data structures in your application.
- Include the TARGET RTOS_interface.h (For example, nucleus_interface.h in case of Nucleus Interface) and os_target.h in the source files.
- Modify the OS Changer init code (see sample provided) and the TARGET RTOS root task of your application appropriately. (For example, Application_Initialize)
- Compile and link your application using appropriate development tools. Resolve all compiler and linker errors.
- Port the underlying low-level drivers to Target OS.
- Load the complete application image to the target system and run the application.
- Review the processor and development system documentation for additional information, including specific details on how to use the compiler, assembler, and linker.

# Error Handling

Applications receive a run-time error via the OS_Fatal_Error() function on some occasions. This happens due to:

- Unsupported API function call, or
- Unsupported parameter value or flag option in a API call, or
- Error occurred on the target OS for which there are no matching error codes in Cross-OS Interface.

OS Changer calls OS_Fatal_Error and passes along an error code and error string. The OS_Fatal_Error handling function is fully customizable to the application needs. At the moment it prints the error message if the OS_DEBUG_INFO conditional compile option is set, then OS_Fatal_Error does not return. For more details on error handling and definition of this function, refer to the Cross-OS Interface Reference Guide. The non-zero value in the error code corresponds to the underlying RTOS API error. Refer to the target OS documentation for a better description of the errors. Error Handling section lists the errors and the reasons for the occurrence.

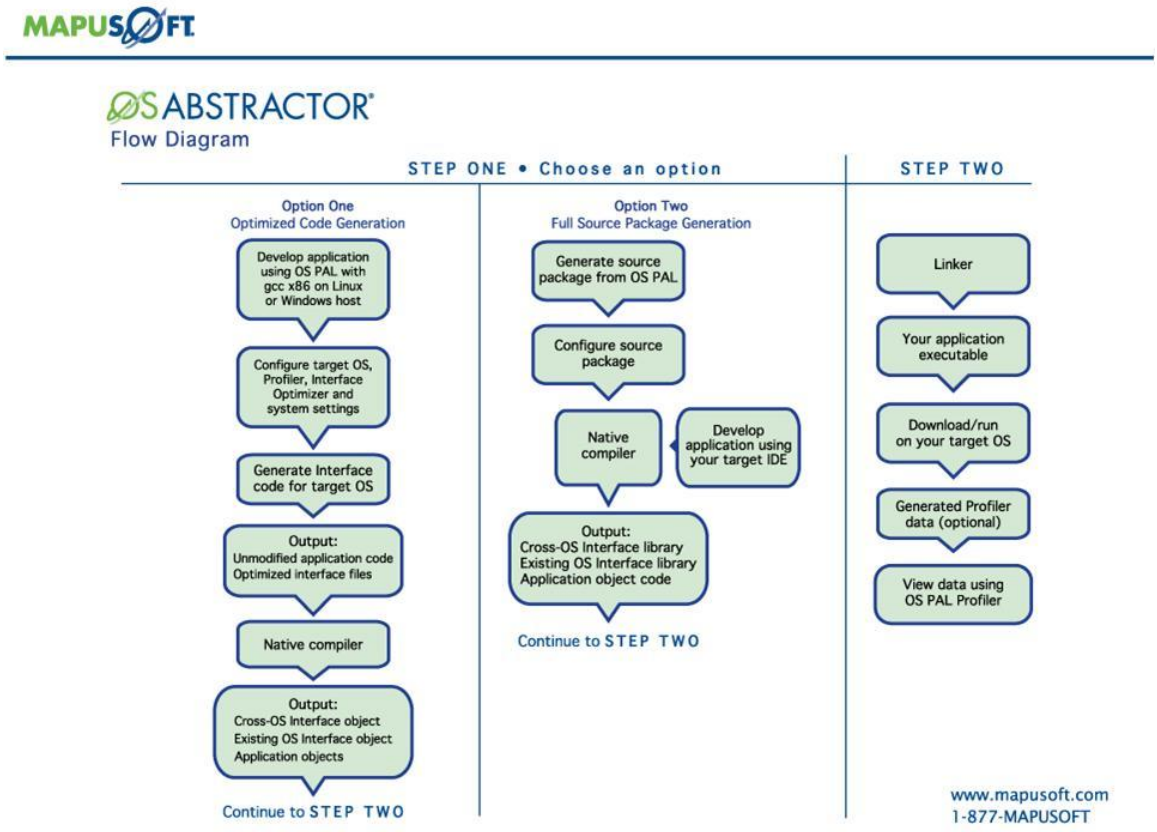
# Chapter 5. OS Abstractor For Developing Portable Software

OS Abstractor is designed for use as a C library. Services used inside your application software are extracted from the OS Abstractor libraries and are then combined with the other application objects to produce the complete image. This image may be downloaded to the target system or placed in ROM on the target system. OS Abstractor will also function under various host environments.

This chapter contains the following topics:

# About OS Abstractor

Developing a solid software architecture that can run on multiple operating systems requires considerable planning, development and testing as well as upfront costs associated with the purchase of various OS and tools to validate your software. MapuSoft's OS Abstractor is an effective and economical software abstraction alternative for your embedded programming. By using OS Abstractor, your embedded application can run on many real time (RTOS) and non-real time operating systems to negate any porting issues in the future when your platform changes.




***OS Abstractor Flow Diagram***

# Interfaces Available for OS Abstractor

The following are the OS Abstractor products:

- POSIX
- micro-ITRON
- VxWorks
- pSOS
- Nucleus
- Windows

Application developers need to specify the target operating system that the application and the libraries are to be built for inside the project build scripts. Application developers can also customize OS Abstractor to include only the components that are needed and exclude the ones that are not required for their application.

If the Application also uses Interface products, additional configuration may be necessary. Please refer to the individual Interface documents.

# Using OS Abstractor

The steps for using OS Abstractor are described in the following generic form:

1. Include os_target.h in all your application source files.
2. Set the appropriate compiler switches within the project build files to indicate the target OS and other target configurations.
3. Configure the pre-processor defines found in the cross_os_usr.h header file under each target OS folder to applications requirements.
4. Initialize the OS Abstractor library by calling OS_Application_Init() function. If you are also using POSIX Interface, then also use OS_Posix_Init() function call to initialize the POSIX component as well. If you use OS Changer(s), you may need to call other appropriate initialization functions as well. After initialization, create your initial application resources and start the application's first task. After this and within the main thread, call OS_Application_Wait_For_End() function to suspend the main thread and wait for application re-start or termination requests.
5. Compile and link your application using appropriate development tools.
6. Download the complete application image to the target system and let it run.

**NOTE**: Make sure to disable User Account Control (UAC) in order to have administration permission in Windows Vista and Windows7.

## Turning Off UAC

In order to run our products successfully, users need to turn off the User Access Control (UAC).

To turn off UAC:

- On **Windows Vista**:
  1. Go to **Start> Control Panel > Security Center > Other Security Settings.**
  2. Turn off **User Access Control**.
- On **Windows 7**:
  1. Go to **Start>Control Panel\User Accounts and Family Safety\User Accounts.**
  2. Set the notification to **Never Notify**.

Refer to the sample demo applications provided with OS Abstractor as a reference point to start your application. Please review the target processor and appropriate development tools documentation for additional information, including specific details on how to use the compiler, assembler, and linker.

# Chapter 6. Full Library Package Generator

MapuSoft enables you to generate a full library code package to create libraries and develop applications using your own IDE. You can manually scale and configure the product by modifying the user configuration file.

**Note:** Before you begin, refer to MapuSoft System Configuration Guide.

This chapter contains the following topics:

Generating Full Library Packages

Generating Binary Packages

# Generating Full Library Packages

**NOTE**: This feature requires a Library Package generation license. Click http://mapusoft.com/contact/ to send a request to receive licenses and documentation.

OS PAL can also create full library packages to complete the porting and development outside of OS PAL with your own tools and environment.

**NOTE:** To generate full library package on Windows Interface, ensure that the flag INCLUDE_OS_PROCESS is set to OS_TRUE in the cross_os_usr.h configuration file.

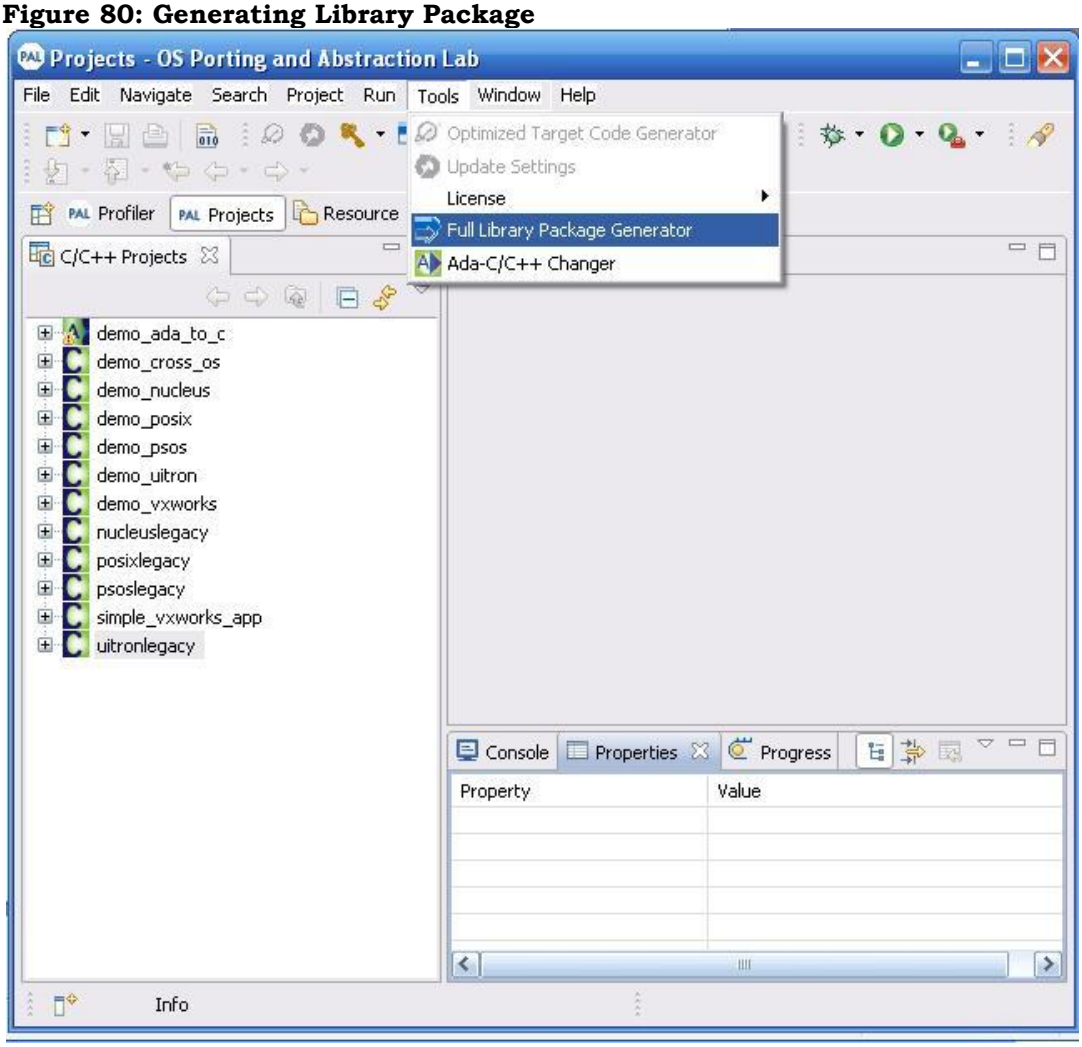
**To generate full source library package, follow the steps**:

## 1. Generating Full Library Package

To generate full library package:

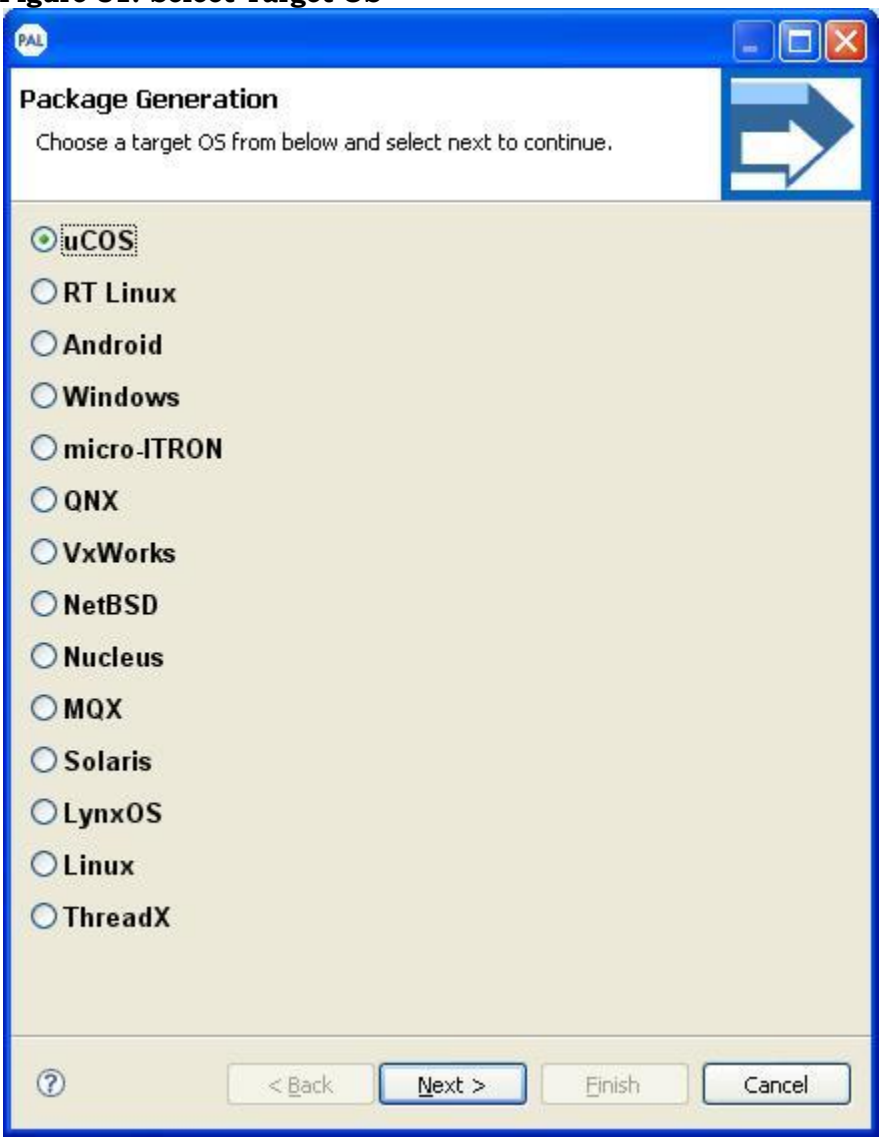
a) From OS PAL main menu, click **Full Library Package Generator** button on the tool bar as highlighted in Figure 80 Or select **Tools > Full Library Package Generator.**

**Figure 80: Generating Library Package**



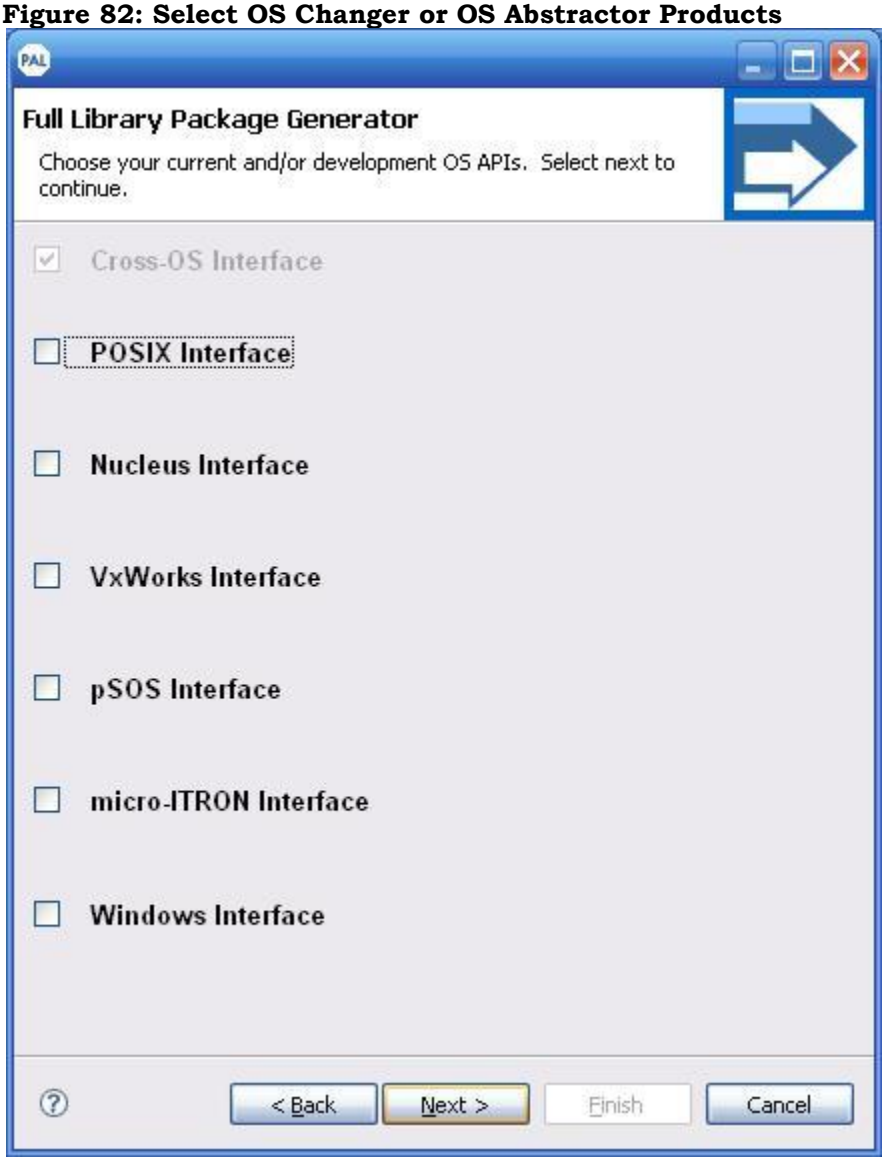
b) On Full Library Package Generator window, select the required Target OS from the list and click **Next** as shown inFigure 81.

**Figure 81: Select Target OS**



c) Select the development OS APIs needed to generatefull library package and click **Next** as shown inFigure 82.

**Figure 82: Select OS Changer or OS Abstractor Products**



d) Select the destination path to save the generated package and click **Finish** as shown in Figure 83.

**Figure 83: Select Destination Path**

e) The successful library package generation is shown inFigure 84.

**Figure 84: Full Library Package Generation Verification Report**

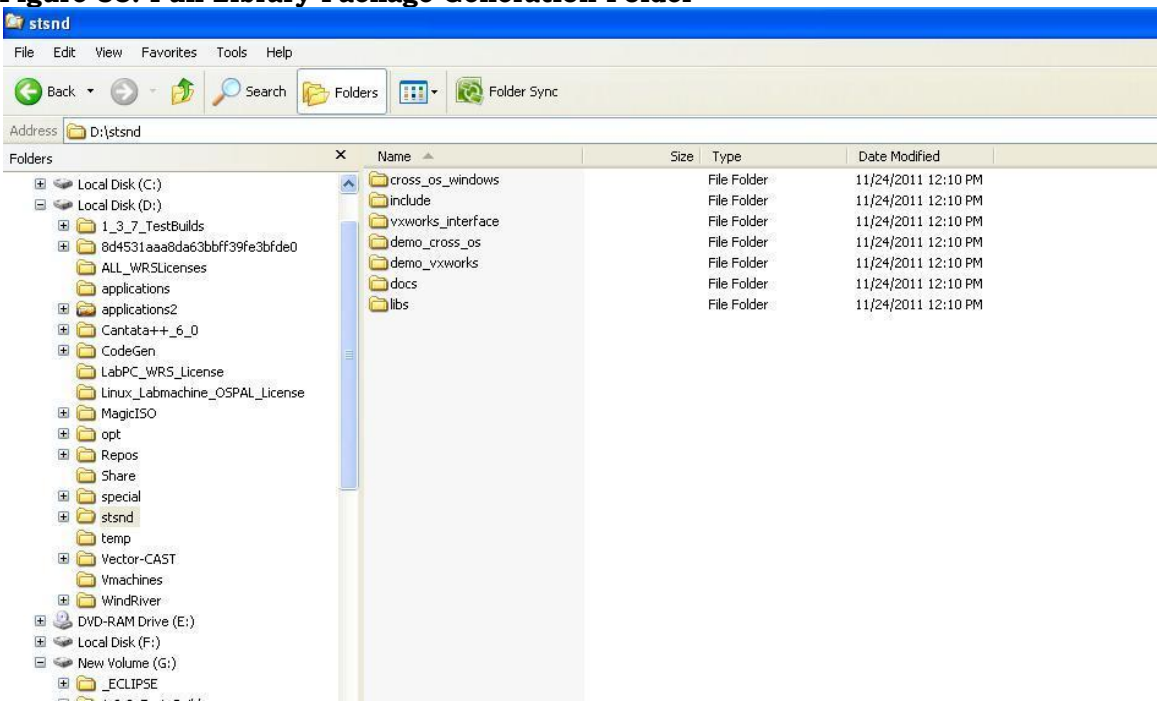
## 2. Show the Full Library Package Created by OS PAL

The full library package includes libraries with full source code to manually link into the applications. Once the application is recompiled with MapuSoft's products it will run on the new OS.

To view the generated full library package:

a) Open the folder created by OS PAL.
b) Click on the folder to view the files included in the package as shown in Figure 85, which include:
   - A sample application
   - Libraries containing the MapuSoft products needed to run their application on the new OS

**Figure 85: Full Library Package Generation Folder**




MapuSoft provides source and libraries in the following folder structure:

The standalone library package will require the header files. The directory structure should be the same as a standalone source package without the source.

```
Mapusoft

Cross_os_windows
            Include
                    <headers>
                 Sources
                         <source file>
                 Lib
                      Windows
                            X86
                                     Vs2008
```

MapuSoft provides six different settings for libraries as shown below:

cross_os_usr.h will use the correct settings for the library build specified using the OS_LIB_BUILD pre-compile flag as described below:

- **OS_LIB_REL_PROC_PROF**—If OS_LIB_BUILD is set to OS_LIB_REL_PROC_PROF as the compiler option, the library linked must be the one built as release with Process mode and Profiling turned on.
- **OS_LIB_REL_PROC_NOPROF**—If OS_LIB_BUILD is set to OS_LIB_REL_PROC_NOPROF as the compiler option, the library linked must be the one built as release with Process mode and Profiling turned off.
- **OS_LIB_REL_NOPROC_PROF**—If OS_LIB_BUILD is set to OS_LIB_REL_NOPROC_PROF as the compiler option, the library linked must be the one built as release without Process mode and Profiling turned on.
- **OS_LIB_REL_NOPROC_NOPROF**—If OS_LIB_BUILD is set to OS_LIB_REL_NOPROC_NOPROF as the compiler option, the library linked must be the one built as release without Process mode and Profiling turned off.
- **OS_LIB_DEBUG_PROC**—If OS_LIB_BUILD is set to OS_LIB_DEBUG_PROC as the compiler option, the library linked must be the one built with Debug mode and Process turned on.
- **OS_LIB_DEBUG_NOPROC**—If OS_LIB_BUILD is set to OS_LIB_DEBUG_NOPROC as the compiler option, the library linked must be the one built with Debug mode and Process turned off.

You must also ensure that the matching library is being linked into your application.OS_LIB_BUILD must be set to the correct value corresponding to the library that is linked. If these do not match, it will not work.

OS_LIB_BUILD is set in the project/makefile settings.

Make sure if your application is built for 32bit architecture, OS_CPU_64BIT is set as OS_FALSE as a command line option.

# Developing Cross-OS Application

The steps for developing Cross-OS applications on host targets are described in the following generic form:

1. Include os_target.h in all your application source files.
2. Set the appropriate compiler switches within the project build files to indicate the target OS and other target configurations
3. Configure the pre-processor defines found in the cross_os_usr.h header file under each target OS folder to applications requirements
4. Initialize the OS Abstractor library by calling OS_Application_Init() function. If you are also using POSIX Interface, then also use OS_Posix_Init() function call to initialize the POSIX component as well. If you use OS Changer(s), you may need to call other appropriate initialization functions as well. After initialization, create your initial application resources and start the application's first task. After this and within the main thread, call OS_Application_Wait_For_End() function to suspend the main thread and wait for application re-start or termination requests.
5. Compile and link your application using appropriate development tools.
6. Download the complete application image to the target system and let it run.

Refer to the sample demo applications provided with OS Abstractor as a reference point to start your application. Please review the target processor and appropriate development tools documentation for additional information, including specific details on how to use the compiler, assembler, and linker.

# Generating Binary Packages

**NOTE**: If you want to build a library as a Shared Library, use the makefile named makefile_s under cross_os_xxxx/specific/x86/gnu/makefile.

# Chapter 7. Optimized Target Code Generator

OS PAL's Optimized Target Code Generator generates porting and Cross-OS interface source code optimized for your application. This allows you to create project files. This also includes the system settings you chose in the GUI-based Wizard.

**Note:** Before you begin, refer to MapuSoft System Configuration Guide.

This chapter contains the following topics:

Generating Optimized Target Code

Generating Project Files for your Target

Inserting Application Code to Run only on Target OS Environment

Inserting Application Code to Run only on Target OS Environment

The user configuration is done by setting up the appropriate value to the pre-processor defines found in the cross_os_usr.h.

**NOTE**: Make sure the OS Abstractor libraries are re-compiled and newly built whenever configuration changes are made to the cross_os_usr.h when you build your application. In order to re-build the library, you would actually require the full-source code product version (not the evaluation version) of OS Abstractor.

Applications can use a different output device as standard output by modifying the appropriate functions defines in os_target_usr.h along with modifying os_setup_serial_port.cmodule if they choose to use the format Input/Output calls provided by the OS Abstractor.

You can add some application code or target specific things such as memory allocations such as Heap Size and Shared memory which are specific to target environments.

## Target OS Selection

Based on the OS you want the application to be built, set the following pre-processor definition in your project setting or make files:

| Flag and Purpose | Available Options |
|---|---|
| **OS_TARGET**<br>To select the target operating system. | The value of the OS_TARGET should be for the Cross-OS Interface product that you have purchased. For Example, if you have purchased the license for :<br>**OS_NUCLEUS** – Nucleus PLUS® from ATI<br>**OS_THREADX** – ThreadX® from Express Logic<br>**OS_VXWORKS** – VxWorks® from Wind River Systems<br>**OS_ECOS** – eCOS standards from Red Hat<br>**OS_MQX** - Precise/MQX® from ARC International<br>**OS_UITRON** – micro-ITRON standard based OS<br>**OS_LINUX** - Open-source/commercial Linux® distributions<br>**OS_WINDOWS** – Windows 2000, Windows XP®, Windows CE, Windows Vista from Microsoft. If you need to use the Cross-OS Interface both under Windows and Windows CE platforms, |

| Flag and Purpose | Available Options |
|---|---|
| | then you will need to purchase additional target license.<br>**OS_TKERNEL** – Japanese T-Kernel® standards based OS<br>**OS_LYNXOS** - LynxOS® from LynuxWorks<br>**OS_QNX** – QNX operating system from QNX<br>**OS_LYNXOS** – LynxOS from LynuxWorks<br>**OS_SOLARIS** – Solaris from SUN Microsystems<br>**OS_ANDROID** – Mobile Operating System running on Linux Kernel<br>**OS_NETBSD** – UNIX like Operating System<br>**OS_UCOS** – UCOS® from Micrium<br>For example, if you want to develop for ThreadX, you will define this flag as follows:<br>OS_TARGET = OS_THREADX<br>PROPRIETARY OS: If you are doing your own porting of Cross-OS Interface to your proprietary OS, you could add your own define for your OS and include the appropriate OS interface files within os_target.h file. MapuSoft can also add custom support and validate the OS Abstraction solution for your proprietary OS platform |

Running OS PAL Generated Code on your Target

# Generating Optimized Target Code

This section describes how to generate optimized target code using OS PAL. Most of the configurations described below can also be changed at run time using the OS_Application_Init function.

**NOTE 1**: This feature requires a target license. Click http://mapusoft.com/contact/ to send a request to receive licenses and documentation.

**NOTE 2:** For all Optimized Target Code Generation the preprocessor OS_HOST flag is set to OS_FALSE. If the user intends to do the host development on the optimized target code, they need to change this preprocessor flag to OS_TRUE manually.

**NOTE 3:** On Linux target, PC hangs while running demo_uitron from terminal if you terminate the execution by Ctrl+C. Make sure that #define OS_BUILD_FOR_SMP is set to False when compiling for non SMP processors.

**NOTE 4**: On Linux target, System hangs while Target Code Optimized application runs. As a workaround, do not terminate the application till profiler is generated. If you terminate in between your PC will hang.

**NOTE 5**: If you select a library project, which either has, a C/C++ generic library project or OS PAL library project, Target Code Generator icon is disabled.

**NOTE 6**: API optimization is not supported for OS PAL libraries linked with application project during target code generation.

To enable **Indexer**:

1. Select**Project > Properties > C/C++ General > Indexer**.
2. Click**Enable Project Specific Settings** check box, and click**Enable Indexer**check box.
3. Click **Apply**and **OK**.

## Optimized Target Code Generation for Ada Projects

OS PAL allows Optimized Target Code Generation for Ada-C/C++ Changer Projects, when the projects are created with OS Abstractor Integration.

For Ada-C/C++ Changer projects created without OS Abstractor Integration, Optimized Target Code Generation should not be done. The generated code WILL NOT work since there is no call to OS_Application_Initialize.

For Ada-PAL Compiler Projects, Cross-OS interfaces are added directly to the project as target sources, if you have a valid and relevant Standalone Package Generator license. If Optimized Target Code Generation is attempted on these projects, all the Cross-OS functionality, being part of application, is again redefined in cross_os.c. This will give re-definition errors on compile time.

Hence Ada-PAL Compiler projects cannot be code-optimized.

**NOTE 1**: The project file generated to QNXMomentics 4.x IDE using optimized target does not enable the build variant, so you need to manually enable the build variant after importing in the QNX IDE.

**To enable the Build Variant**:

- Select the project and go to **Project Properties->C/C++ General->QNX C/C++Project.**

- Select **Build variant** tab.
- To enable the build variant, select the **X86** check box.

**NOTE 2:** Check if the Indexer is enabled**.** Generating Optimized Target Code will not work if Indexer is OFF.

**NOTE 3:**For all Optimized Target Code Generation the preprocessor OS_HOST flag is set to OS_FALSE. If the user intends to do the host development on the optimized target code, they need to change this preprocessor flag to OS_TRUE manually.

**NOTE 4**: The QNX Momentics IDE has an issue where relative path names are not updated unless there is a modification to the project settings. This will cause the initial build of the Full Source version of OS Abstractor to fail since the project files were created in a different location than where they were installed.

**To force Momentics to update these paths**:

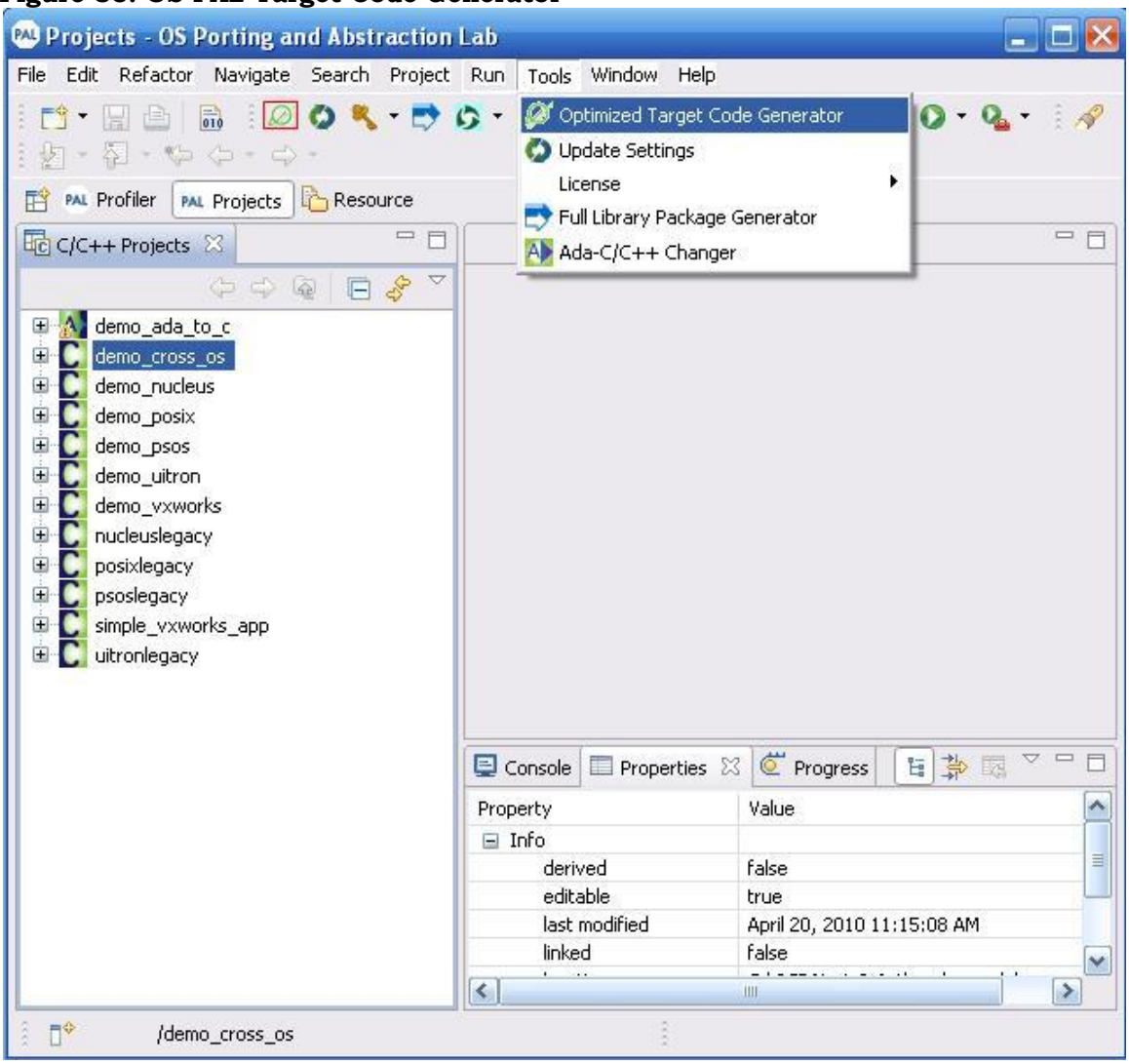
- Right-click on the project and select **Properties** from the context menu. Then click **Apply** and close the properties window.

**NOTE 5:** To generate optimized target code on Windows Interface, ensure that the flag INCLUDE_OS_PROCESS is set to OS_TRUE in the cross_os_usr.h configuration file.

**To generate Optimized Target Code**:

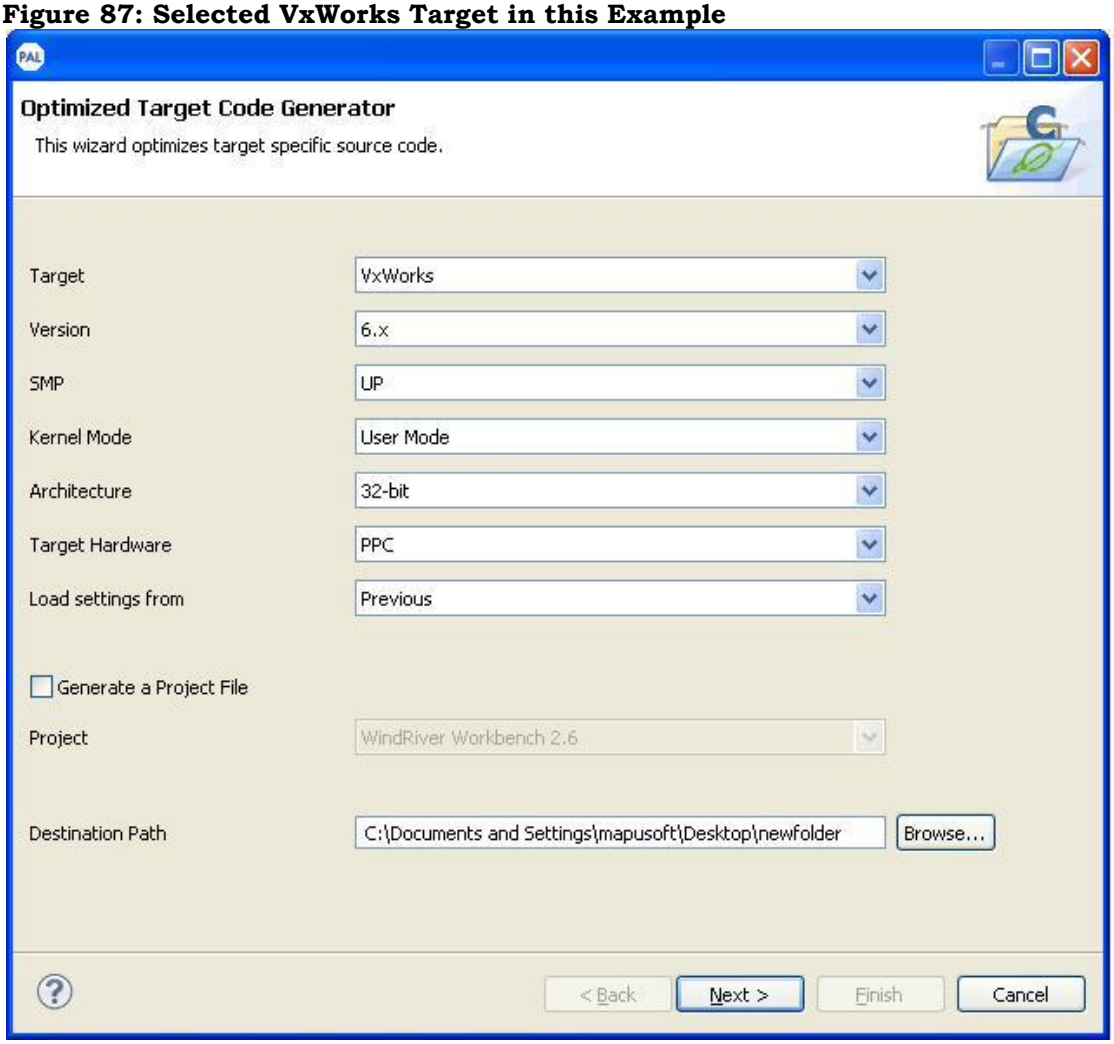
1. From OS PAL projects, select a project.
2. From OS PAL main menu, click **Tools > Optimize Target** or click on the OS PAL Optimized Target Code Generator button  as shown in Figure 86.

**Figure 86: OS PAL Target Code Generator**



3. From OS PAL Optimized Target Code Generator window, select your target platform specifications from the drop down list. VxWorks operating system is selected as an example as shown in Figure 87.

**Figure 87: Selected VxWorks Target in this Example**

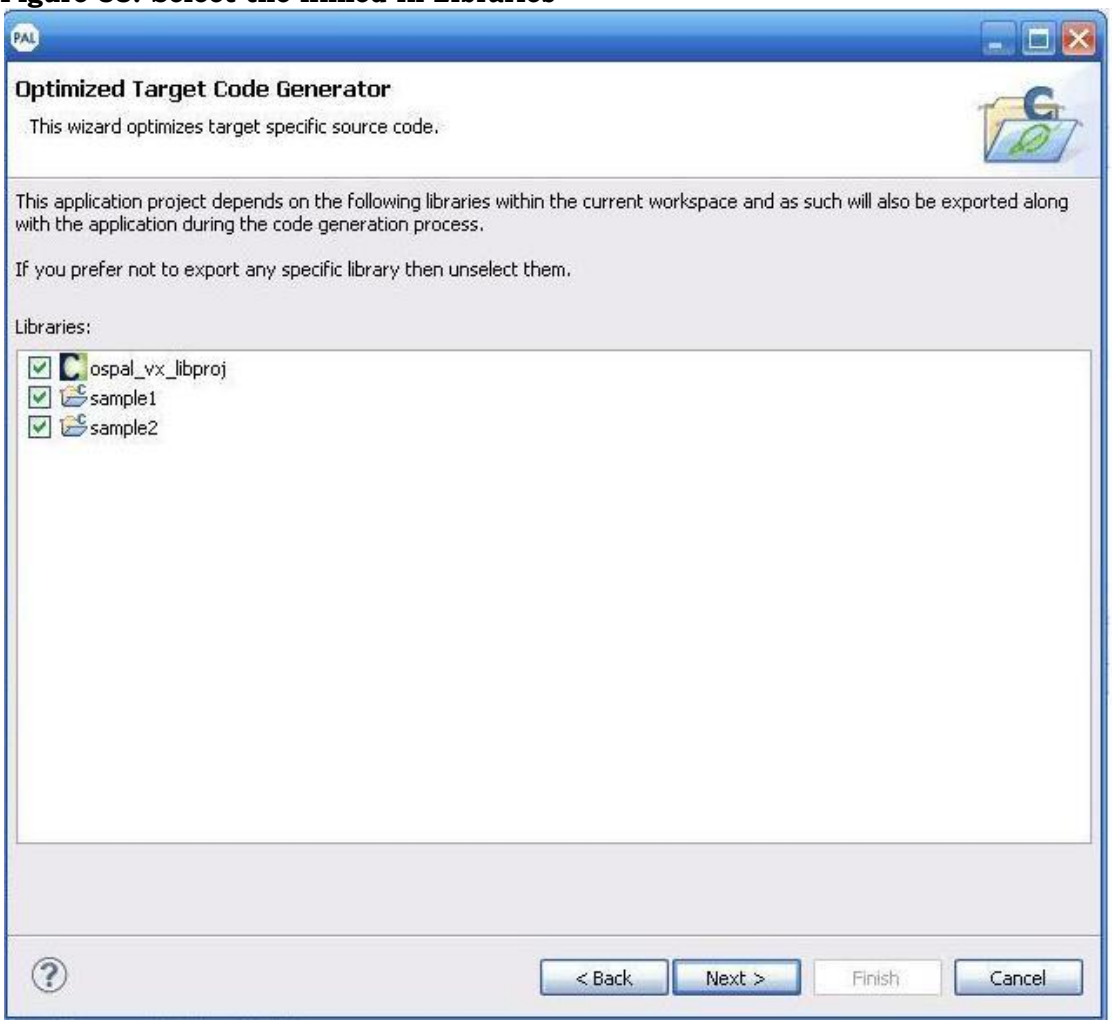


The field descriptions on OS PAL Optimized Target Code Generator window are as follows:

| Field | Description | Your Action |
|---|---|---|
| Target | Specifies the target OS name. | Enter the target OS name in the text box. |
| Version | Based on the Target OS name you selected, this specifies, the available version names listed in the Version drop down list. | Select the appropriate target version. |
| SMP | This specifies the Symmetric Multi processor. | Select this if your target supports SMP or UP. |
| Kernel Mode | If applicable to the Target OS name and version, this specifies the following modes:<br>• User mode<br>• Kernel mode | Select the Target kernel mode by selecting it from the drop down list. |
| Architecture | This specifies the architecture of the Target OS. The options are:<br>• 32-bit<br>• 64-bit | Select the architecture you need. |
| Target Hardware | Specifies the type of target hardware used to complete code optimization.<br>**Note**: You can select the target hardware only when you select VxWorks as your target OS. | Select the type of target hardware used. The options are: PPC,PPC_604,X86,ARM, M68K,MCORE,MIPS,SH, SIMLINUX,SIMNT,SIMSOLARIS,SPARC |
| Load Settings | This specifies the following two options to load settings from:<br>• **Previous**: If you select Previous, then initial values for this wizard are loaded from previously saved settings and populated.<br>• **Default:** If you select Default, then the values from default settings are populated. | Select the option to load settings from by selecting from the drop down list. |
| Generate a Project File | Specifies if you want to generate a project file. | Select the check box next to Generate project file. |
| Project | Specifies the different target project types that you can generate for this project. The generated project files are directly imported into the specified IDE (Eclipse/Visual Studio), and this project becomes a project of that IDE. | Select the project from the drop down list. |
| Destination Path | Specifies the path to place the generated optimized target code. | Click **Browse** and select the folderto place the generated code. |

4. If you select an application project, OS PAL checks if the application project has any linked-in libraries and queries the workspace for any matching library projects. If OS PAL finds any libraries, then it will listall the available libraries on the page as shown inFigure 88.All the libraries are selected by default, and you can select or deselect any/all libraries to export library sources along with the application during code generation process.Select the libraries and click **Next**.

**Figure 88: Select the linked-in Libraries**



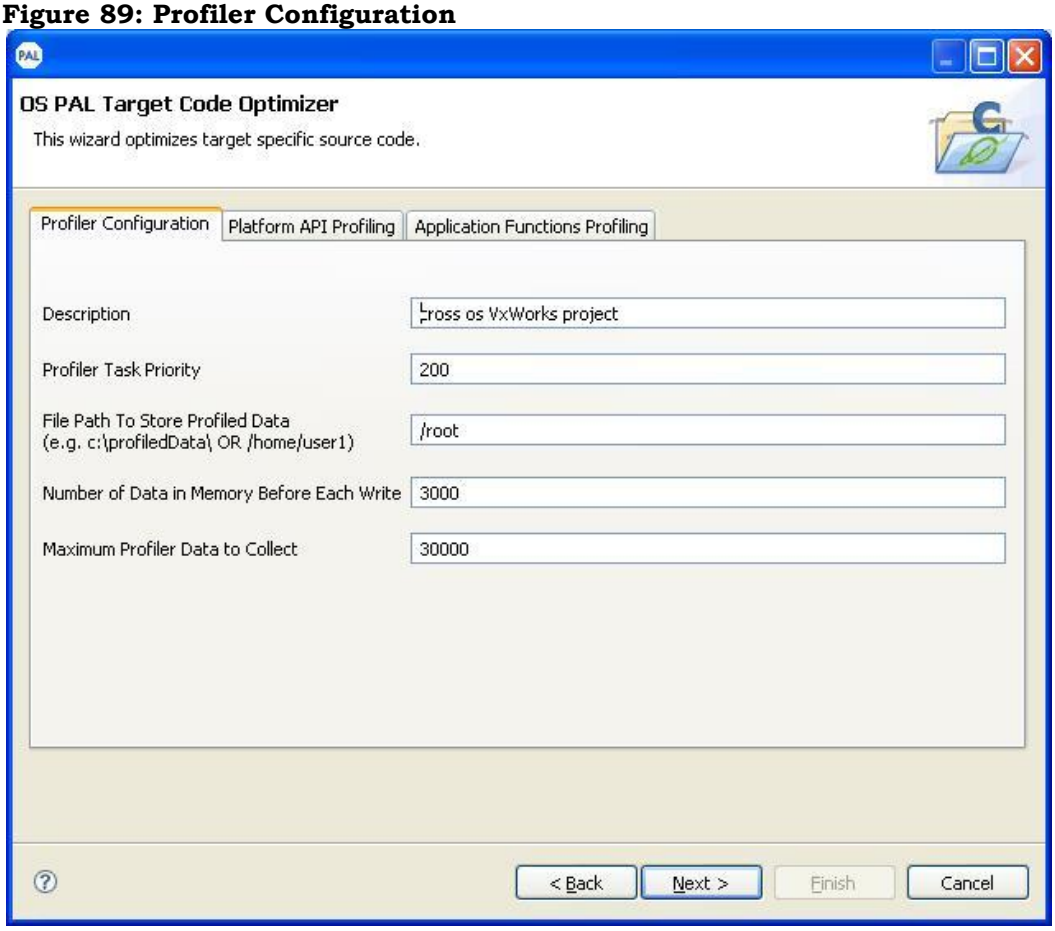

**NOTE**: When you export OSPAL libraries, it will skip API Optimization.

Target Code Generator will contain the following folders/files:

- Application project sources and project/make files.
- OS Abstractor Interfaces (the ones that are included in the project)
- Library sources without project files.

5. On **Profiler Configuration** tab, define your profiler data specifications as shown in Figure 89.

**Figure 89: Profiler Configuration**

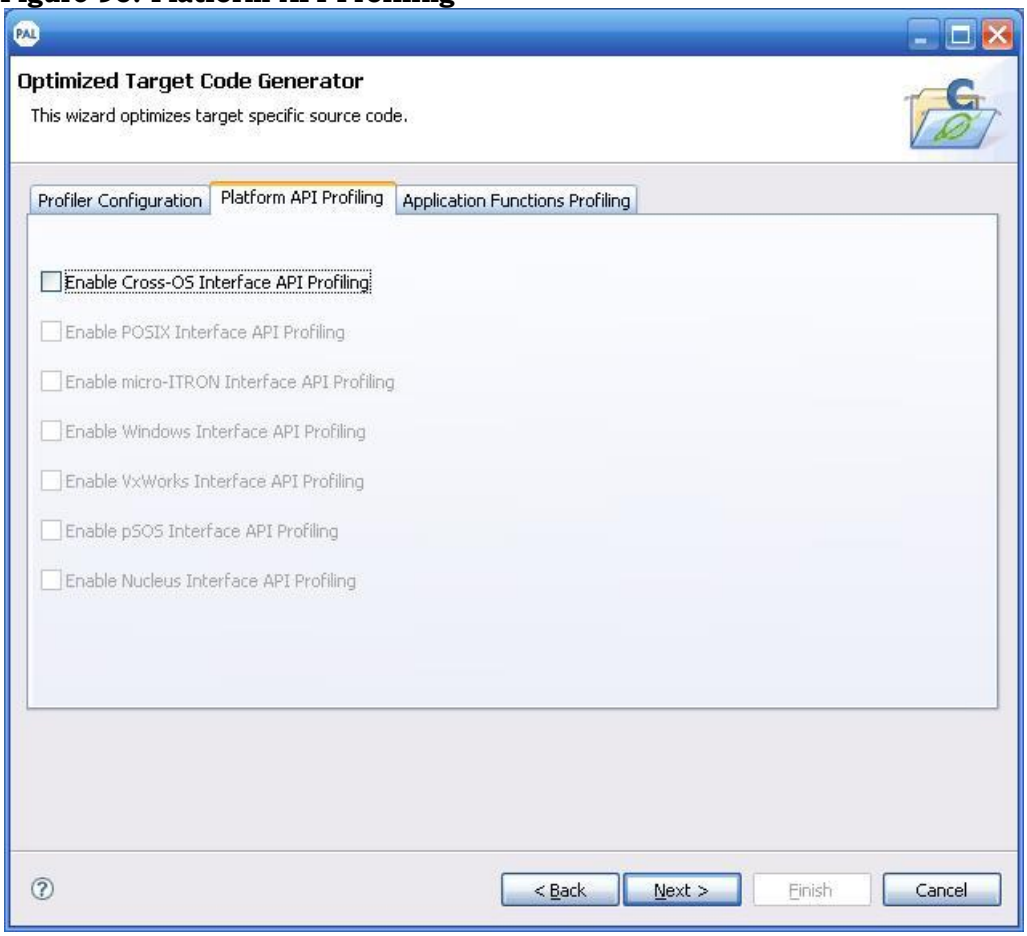


The field descriptions on Profiler Configuration tab are as follows:

| Field | Description | Your Action |
|---|---|---|
| Description | Specifies the description for the Cross-OS Interface project. | Type description for the Cross-OS Interface project. |
| Profiler Task Priority | Specifies the priority level of the profiler thread. | Enter a priority level for the profiler thread.<br>The value can be between 0 through 225. The default value is set to 200. |
| File Path to Store Profiled Data | Specifies the directory location where the profiler file will be created. | Enter a data file path.<br>The default location set is */root* on Unix based machines and *c/* on MS Windows machine. |
| Number of Data in Memory Before Each Write | Specifies the depth of the profiler queue. | Enter the number of data in memory before each write.<br>The default value is set to 3000. |
| Maximum Profiler Data to Collect | Specifies the maximum records collected in the XML file. | Enter the number of profiler messages.<br>The default value is set to 30000. |

6. On **Platform API Profiling** tab, select the check box to enable your appropriate Interface API Profiling as shown in Figure 90.

**Figure 90: Platform API Profiling**

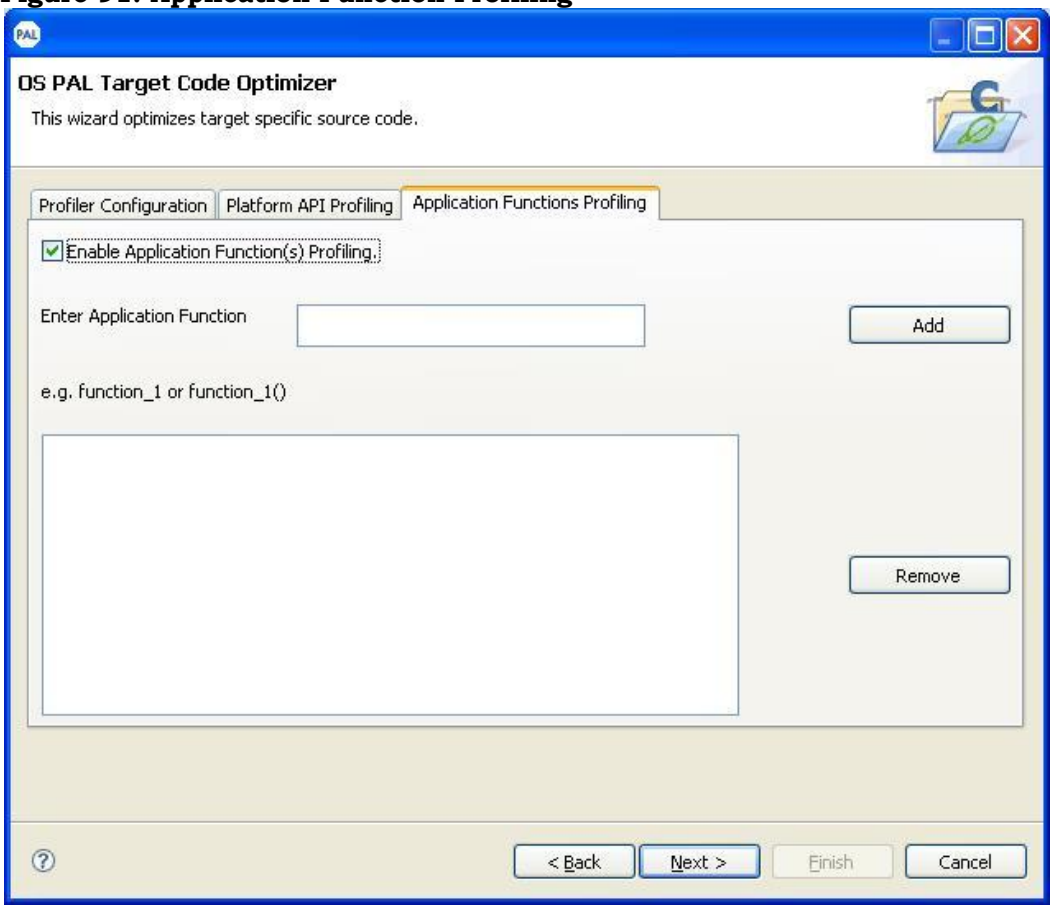


The field descriptions on Platform API Profiling tab are as follows:

| Field | Description | Your Action |
|---|---|---|
| Enable Cross-OS Interface API Profiling | Specifies if the Cross-OS Interface API Profiling feature is enabled or disabled. | Select the check box to enable platform profiling.<br>Platform Profiling means Cross-OS Interface APIs profiling.<br>**NOTE**: By default, Cross-OS Interface API profiling is enabled for all projects. |
| Enable POSIX Interface API Profiling | Specifies if POSIX Interface API Profiling feature is enabled for your project. | Select the check box, if you need profiling for your POSIX APIs. |
| Enable micro-ITRON Interface API Profiling | Specifies if micro-ITRON Interface API Profiling feature is enabled for your project. | Select the check box, if you need profiling for your micro-ITRON APIs. |
| Enable Windows Interface API Profiling | Specifies if Windows Interface API Profiling feature is enabled for your project. | Select the check box, if you need profiling for your Windows APIs. |
| Enable VxWorksInterface API Profiling | Specifies if VxWorksinterface API Profiling feature is enabled for your project. | Select the check box, if you need profiling for your VxWorksInterface APIs. |
| Enable pSOSInterface API Profiling | Specifies if pSOSInterface API Profiling feature is enabled for your project. | Select the check box, if you need profiling for your pSOSInterface APIs. |
| Enable Nucleus Interface API Profiling | Specifies if Nucleus Interface API Profiling feature is enabled for your project. | Select the check box, if you need profiling for your Nucleus Interface APIs. |

7. On **Application Functions Profiling** tab, you can also perform profiling for your specific APIs as shown in Figure 91.

**Figure 91: Application Function Profiling**



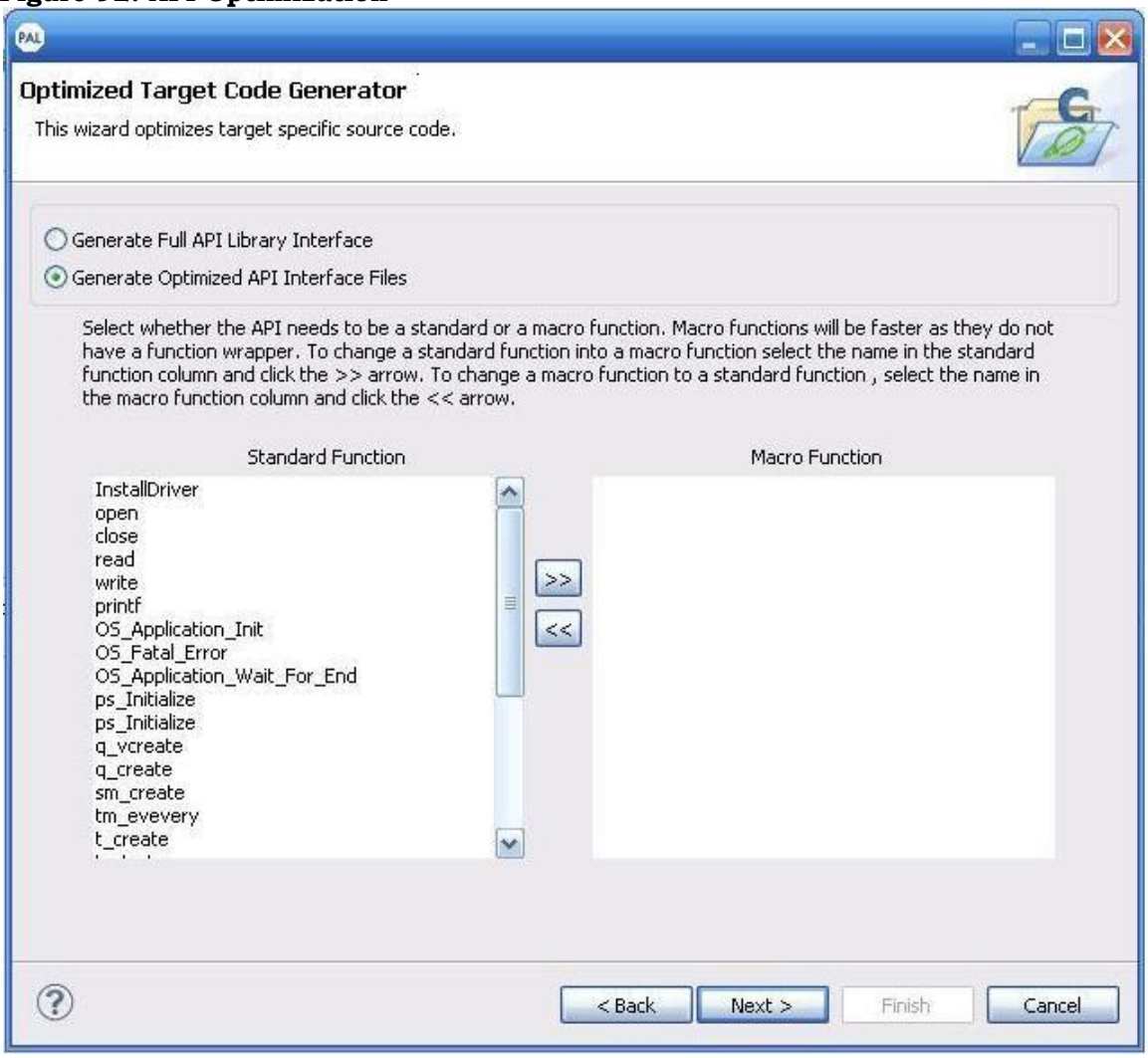
The field descriptions on Application Functions Profiling tab are as follows:

| Field | Description | Your Action |
|---|---|---|
| Enable Application Functions Profiling | Specifies if the Application Functions Profiling feature is enabled or disabled. | Select the check box to enable Application Functions profiling. This profiling is used for User APIs profiling. |
| Enter Application Function | Specifies the name of the Application Function for profiling. | Enter the name of the application function. **NOTE**:This field is case sensitive. |
| Add | Specifies if you want to add any application functions. | To add any application function, enter the name in the text box, and click **Add.** |
| Remove | Specifies if you want to remove any application functions from the list. | To remove any application function from the list, select the name of the application function in the text box, and click **Remove.** |

8. Add your APIs by typing in the name of the API next to **Enter Application Function** text box and click **Add** and click **Next**.

9. On **API Optimization** tab, you can select either to generate Full API Library Interface or Optimized API interface. In API Optimization, you can select the API's which needs to be a standard function or a macro function and move the selected API using the double arrow button as shown in Figure 92. Macro functions will execute faster but will increase the memory footprint of the application and click **Next**.

**Figure 92: API Optimization**




**Need for Code Optimization:** Macro function is used to eliminate the time overhead when a function is called. It is typically used for functions that execute frequently. It also has a space benefit for very small functions, and is an enabling transformation for other optimizations.

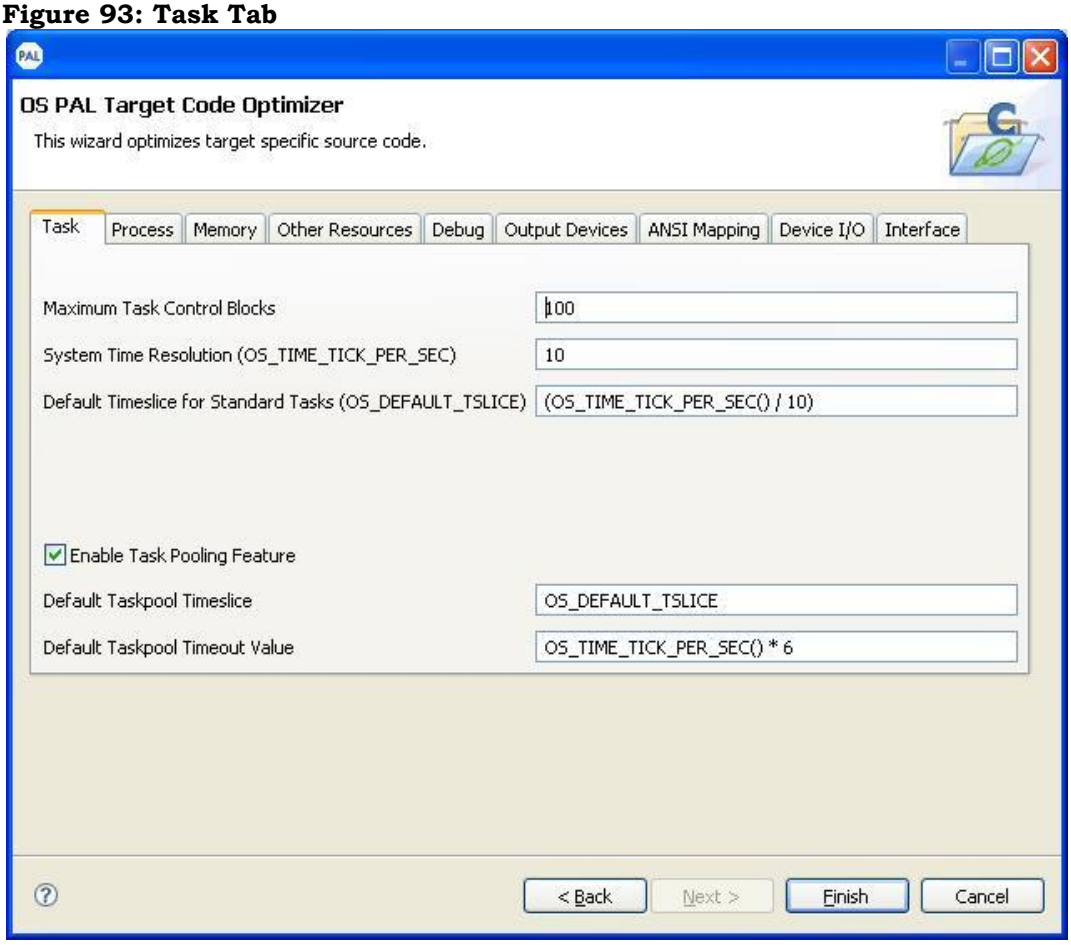
Without macro functions, however, the compiler decides which functions to inline. The programmer has little or no control over which functions are macro functions and which are not. Giving this degree of control to the programmer allows her/him to use application-specific knowledge in choosing which functions to macro.

The field descriptions on API Optimization tab are as follows:

| Field | Description | Your Action |
|---|---|---|
| Generate Full API Library Interface | Specifies if you want to generate full API library package.<br>**Note:** You can do this if you have a valid license for standalone generation. | Select the radio button to generate full library package.<br>**Note:** If you select this option, the rest of the fields on this window are disabled. |
| Generate Optimized API Interface Files | Specifies if you want to generate optimized API interface files.<br>**Note:** If the application includes OS PAL based application libraries, generating optimized API interface option is disabled. | Select the radio button to generate optimized API interface files.<br>**Note:** By default this option is enabled. |
| Standard Function | Specifies if the APIs used in your application are standard functions. | Select the functions used in this application as standard functions for the target OS project.<br>You can select multiple function names at once to place them in the other list.<br>You can select all function names in a list using the selectAll (Ctrl+A) action also. |
| Macro Function | Specifies that a compiler inserts the complete body of the function in every place in the code where that function is used. It is used to eliminate the time overhead when a function is called and execute it frequently. | To select a standard function into a macro function, select the API and click the right arrow.<br>To select a macro function into a standard function, select the API under macro function, and click the left arrow.<br>**Note:** You can use optimization for this. If a function is being called repeatedly, they can improve the performance by making this a macro function. |

10. On **Task** configuration tab, configure the options to your specifications as shown in Figure 93. Applications can create Cross-OS Interface tasks during initialization and will be able to re-use the task envelope repeatedly by selecting the check box next to **Enable Task Pooling Feature**.

**Figure 93: Task Tab**




**NOTE**: In the current release, Task Pooling feature is not supported in ThreadX and Nucleus targets.

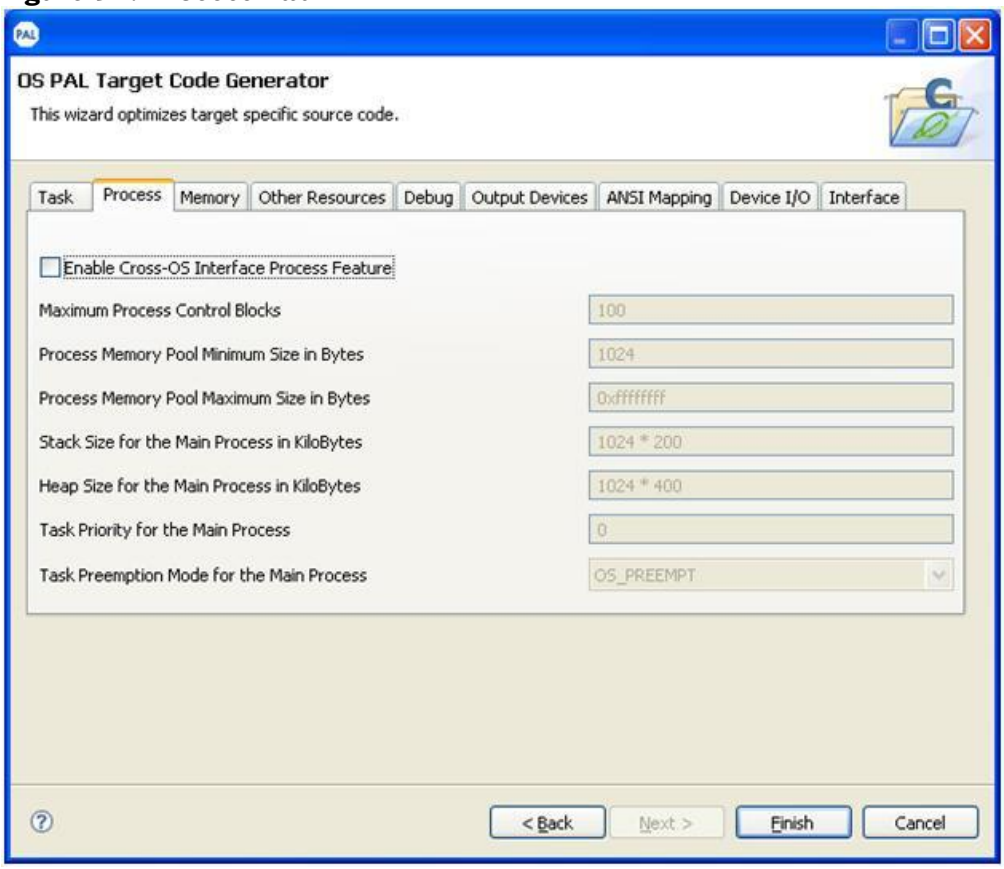
The field descriptions on Task tab are as follows:

| Field | Description | Your Action |
|---|---|---|
| Maximum Task Control Blocks | Specifies the total number of tasks required by the application. | Enter a value.<br>**NOTE:** The default value is 100. One control block will be used by theOS_Application_Init function when the INCLUDE_OS_PROCESS option is true. |
| System Time Resolution (OS_TIME_TICK_PER_SEC) | Specifies the system clock ticks (not hardware clock tick).<br>For example, when you call OS_Task_Sleep(5), you are suspending task for a period (5* OS_TIME_RESOLUTION). | Enter a value.<br>**NOTE:** The default value is 10000 micro second (= 10milli sec).<br>This value is derived from the target OS. If you cannot derivethe value, refer to the target OS reference manual and set thecorrect per clock tick value.<br>**NOTE:** Since the system clock tick resolution may vary across different OS under different target, it is recommended that the applicationuse the macro OS_TIME_TICK_PER_SEC to derive the timing requirementinstead of using the raw system tick value in order to keep the application portable across multiple OS. |
| Default Timeslice for Standard Tasks (OS_DEFAULT_TSLICE) | Specifies the default time slice scheduling window width among the same priority pre-emptable threads when they are all in ready state. | Enter a default timeslice for standard tasks.<br>**NOTE**: The default value is 10 ms. If system tick is 10ms, then the threads will be scheduled round-robin at the rate of every 100ms.<br>**NOTE**: On Linux operating system, the time slice cannot be modified per thread. Cross-OS Interface ignores this setting and only uses the system default time slice configured for the Linux kernel. |
| Enable Task Pooling Feature | Specifies if the Task pooling feature is enabled for this application.<br>Task pooling feature enhances the performances and reliability of application. If you enable the task pooling feature, applications can create Cross-OS Interface tasks during initialization and be able to re-use the task envelope repeatedly. To configure task-pooling, set the following pre-processor flag as follows: INCLUDE_OS_TASK_PO | To enable task pooling feature, select the check box. |

| Field | Description | Your Action |
|---|---|---|
| | OLING. | |
| Default TaskpoolTimeslice | Specifies the default TaskpoolTimeslice. | Enter the default TaskpoolTimeslice.<br>**NOTE**: The default value is OS_DEFAULT_TSLICE. |
| Default Taskpool Timeout Value | Specifies the default Taskpool timeout value. | Enter the default Taskpool timeout value.<br>**NOTE**: The default value is OS_TIME_TICK_PER_SEC() * 6. |

11. On **Process** configuration tab, configure the options to your specifications as shown inFigure 94. Select the check box next to **Enable Cross-OS Interface Process Feature** to allocate the memory from a shared memory region to allow applications to communicate across multiple processes. By disabling this option, the memory will be allocated from the individual application/process specific pool, which is created during the OS_Application_Init function call.

**Figure 94: Process Tab**

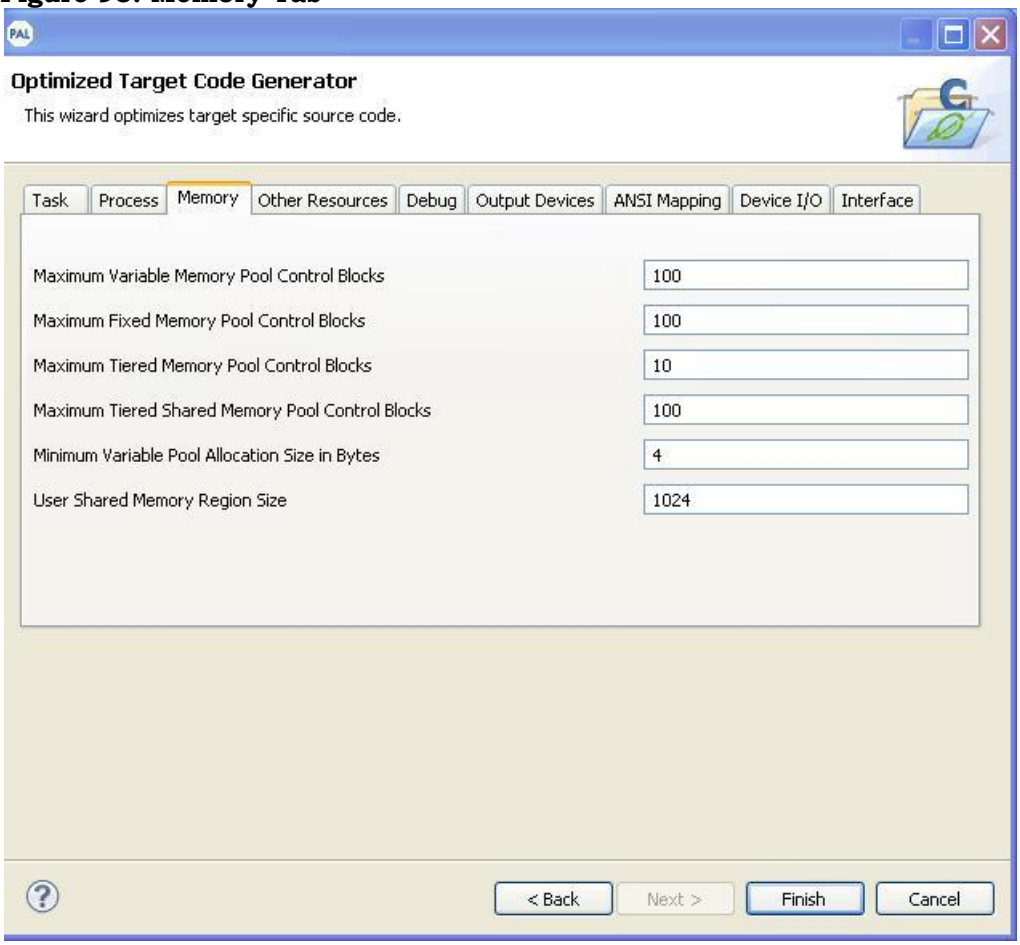


The field descriptions on Process tab are as follows:

| Field | Description | Your Action |
|---|---|---|
| Enable Cross-OS Interface Process Feature | Specifies if the Cross-OS Interface process feature is enabled or disabled. | Select the check box to enable this feature. |
| Maximum Process Control Blocks | Specifies the total number of processes required by the application | Enter the maximum number of process control blocks for the application. **NOTE**: Default value is 100. |
| Process Memory Pool Minimum Size in Bytes | Specifies the minimum size of the process memory pool in Bytes. | Enter the minimum size of the process memory pool. **NOTE**: Default value is 1024 Bytes. |
| Process Memory Pool Maximum Size in Bytes | Specifies the maximum size of the process memory pool in Bytes. | Enter the maximum size of the process memory pool. **NOTE**: Default value is 0xffffffff Bytes. |
| Stack Size for the Main Process in kilobytes | Specifies the stack size for the main process in Kilobytes. | Enter the stack size for the main process. **NOTE**: Default value is 1024 * 200 Kilobytes. |
| Heap Size for the Main Process in kilobytes | Specifies the heap size for the main process in Kilobytes. | Enter the heap size for the main process. **NOTE**: Default value is 1024 * 400 Kilobytes. |
| Task priority for the Main Process | Specifies the task priority for the main process. | Enter the task priority for the mail process. **NOTE**: Default value is 0. |
| Task Preemption Mode for the Main Process | Specifies the preemption status of this task. | Enter the task preemption status of the task. **NOTE**: The valid parameters are: ▪ OS_PREEMPT – Task can be pre-empted by the system. ▪ OS_NO_PREEMPT – Task cannot be pre-empted. |

12. On **Memory** configuration tab, configure the options to your specifications as shown in Figure 95.

**Figure 95: Memory Tab**

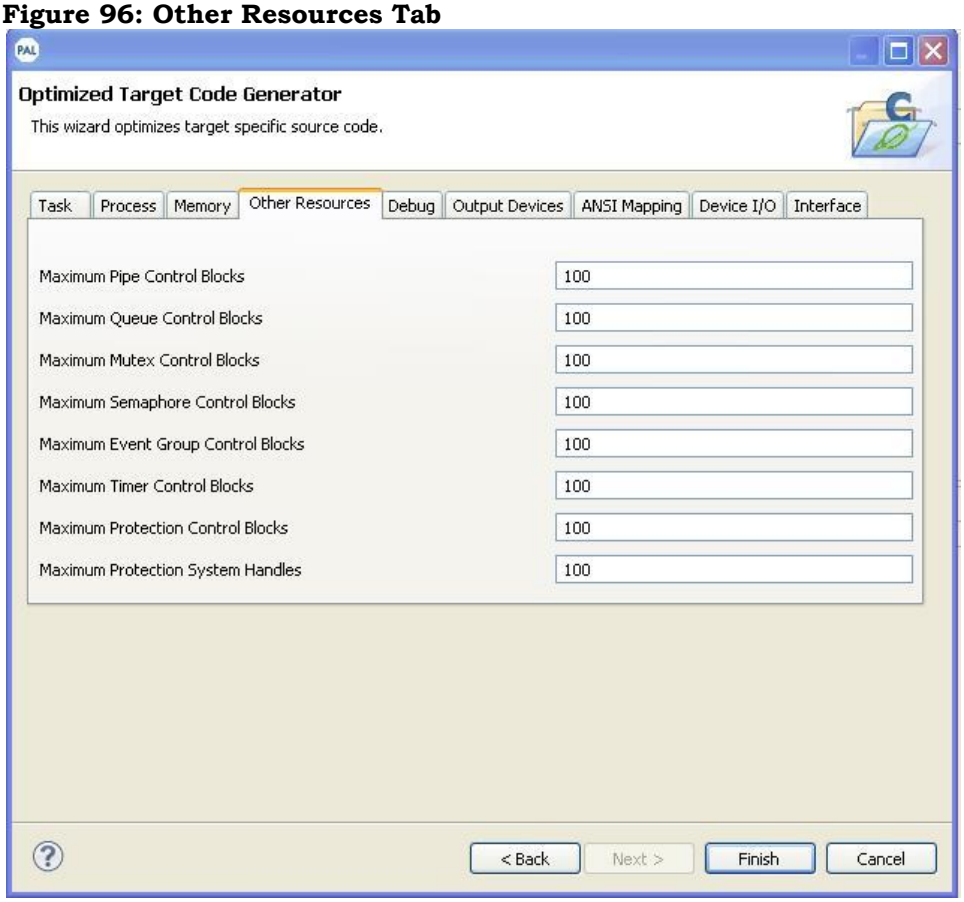


The field descriptions on Memory tab are as follows:

| Field | Description | Your Action |
|---|---|---|
| Maximum Variable Memory Pool Control Blocks | Specifies the total number of dynamic variable memory pools required by the application. | Enter the maximum number of dynamic variable pools. **NOTE**: Default value is 100. |
| Maximum Fixed Memory Pool Control Blocks | Specifies the total number of partitioned (fixed-size) memory pools required by the application. | Enter the maximum number of partitioned memory pools. **NOTE**: Default value is 100. |
| Minimum Variable Pool Allocation Size in Bytes | Specifies the minimum memory allocated by *the malloc()* and/or OS_Allocate_Memory() calls. **NOTE**: Increasing this value further reduces memory fragmentation at the cost of more wasted memory. | Enter the minimum memory allocated. **NOTE**: Default value is 4. Increasing this value further reduces memory fragmentation at the cost of more wasted memory. |
| User Shared Memory Region Size | Specifies the application defined shared memory region usable across all Cross-OS Interface processes/applications. | Enter the user shared memory region size. **NOTE**: Default value is 1024 Bytes. |
| Maximum Tiered Memory Pool Control Blocks | Specifies the total number of Tiered Memory Pools required by the application. | Enter the maximum number of Tiered Memory variable pools. **NOTE**: Default value is 100. |
| Maximum Tiered Shared Memory Pool Control Blocks | Specifies the total number of Tiered Shared Memory Pools required by the application. | Enter the maximum number of Tiered Shared Memory variable pools. **NOTE**: Default value is 100. |

13. On **Other Resources** configuration tab, configure the options to your specifications as shown in Figure 96.

**Figure 96: Other Resources Tab**

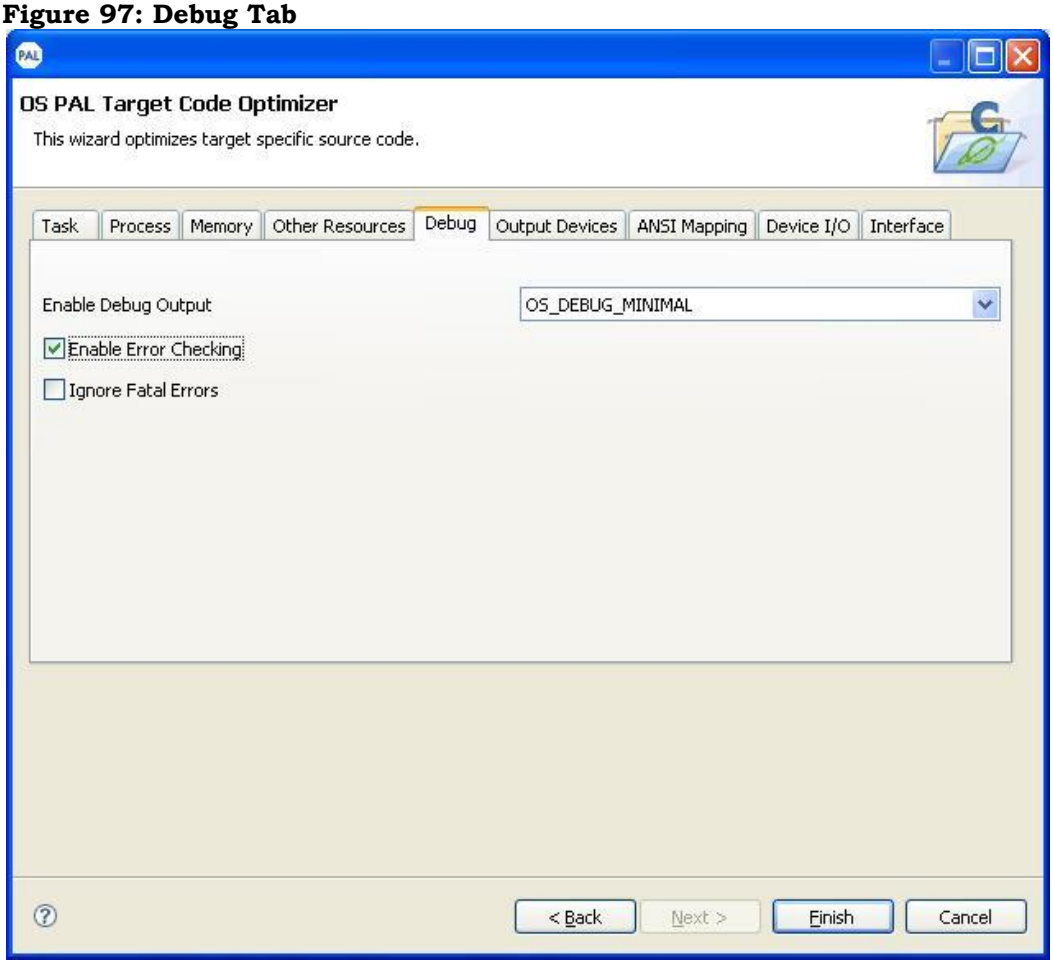


The field descriptions on Other Resources tab are as follows:

| Field | Description | Your Action |
|---|---|---|
| Maximum Pipe Control Blocks | Specifies the total number of pipes for message passing required by the application. | Enter the maximum number of pipe control blocks.<br>**NOTE**: Default value is 100. |
| Maximum Queue Control Blocks | Specifies the total number of queues for message passing required by the application. | Enter the maximum number of queue control blocks.<br>**NOTE**: Default value is 100. |
| Maximum Mutex Control Blocks | Specifies the total number of mutex semaphores required by the application. | Enter the maximum number of mutex control blocks.<br>**NOTE**: Default value is 100. |
| Maximum Semaphore Control Blocks | Specifies the total number of regular (binary/count) semaphores required by the application. | Enter the maximum number of semaphore control blocks.<br>**NOTE**: Default value is 100. |
| Maximum Event Group Control Blocks | Specifies the total number of event groups required by the application | Enter the maximum number of event group control blocks.<br>**NOTE**: Default value is 100. |
| Maximum Timer Control Blocks | Specifies the total number of application timers required by the application | Enter the maximum number of timer control blocks.<br>**NOTE**: Default value is 100. |
| Maximum Protection Control Blocks | Specifies the total number of Protection Control blocks required by the application | Enter the maximum number of Protection control blocks.<br>Note: Default value is 100. |
| Maximum Protection System Handles | Specifies the total number of System handles required by the application | Enter the maximum number of System Handles.<br>Note: Default value is 100. |

14. On **Debug** configuration tab, configure the options to your specifications as shown in Figure 97. The application will be checked for API usage errors by selecting the check box next to **Enable Error Checking**. Disabling error checking will increase the application performance and reduce your code size.

**Figure 97: Debug Tab**

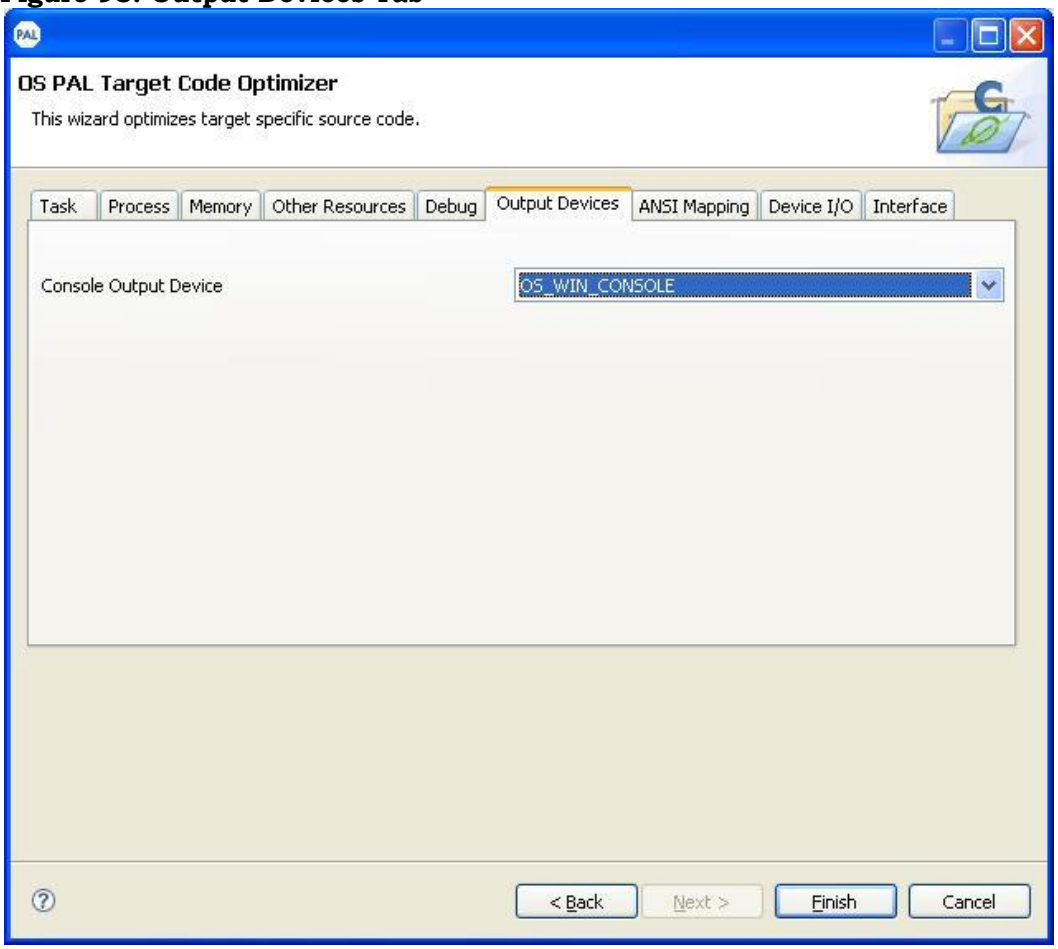


The field descriptions on Debug tab are as follows:

| Field | Description | Your Action |
|---|---|---|
| Enable Debug Output | Specifies if you want to enable the debug output. | Select the debug output from the dropdown menu:<br>▪ OS_DEBUG_VERBOSE – print debug info, fatal and compliance errors<br>▪ OS_DEBUG_MINIMUM – print minimum amount of debug info OS_DEBUG_VERBOSE<br>**Note**: The default value is OS_DEBUG_VERBOSE |
| Enable Error Checking | Specifies if you want to enable the error checking. | To enable error checking, select the check box. Use this option to increase performance and reduce code size.<br>**Note**: By default this feature is enabled. |
| Ignore Fatal Errors | Specifies if you want to enable the feature to ignore fatal errors. | To enable the feature to ignore fatal errors, select the check box.<br>**Note**: By default this feature is disabled. |

15. On **Output Devices** configuration tab, select your output device from the drop drown list as shown in Figure 98.

**Figure 98: Output Devices Tab**

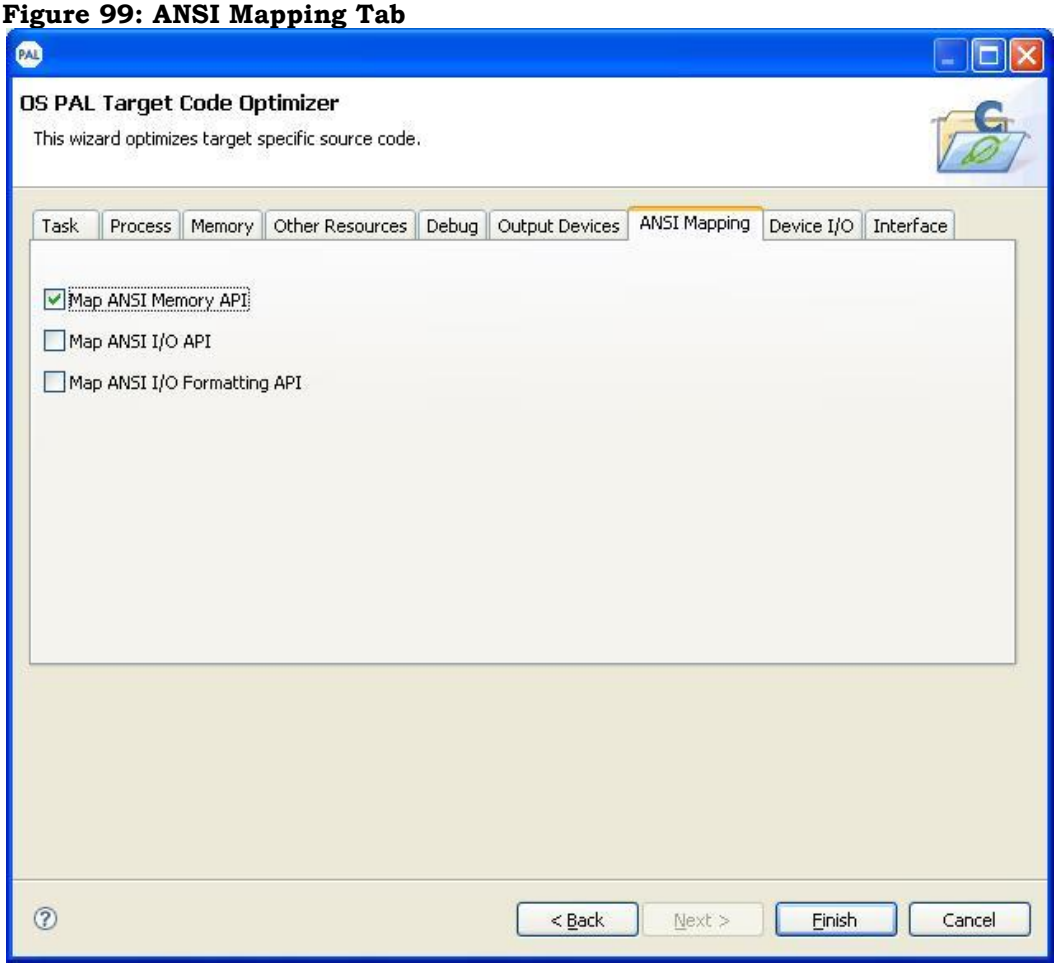


The field descriptions on Output Devices tab are as follows:

| Field | Description | Your Action |
|---|---|---|
| Console Output Device | Specifies the console output device for the application. | Select the output device from the dropdown menu:<br>▪ OS_WIN_CONSOLE – print to console<br>▪ OS_SERIAL_PORT – print to serial<br>**NOTE**: The default value is OS_WIN_CONSOLE<br>User can print to other devices by modifying the appropriate functions within *usr.h* and use Cross-OS Interface's format I/O calls. |

16. On **ANSI Mapping** configuration tab, choose whether to use ANSI mapping as shown in Figure 99. If the ANSI mapping is checked, the application will use MapuSoft's malloc.

**Figure 99: ANSI Mapping Tab**

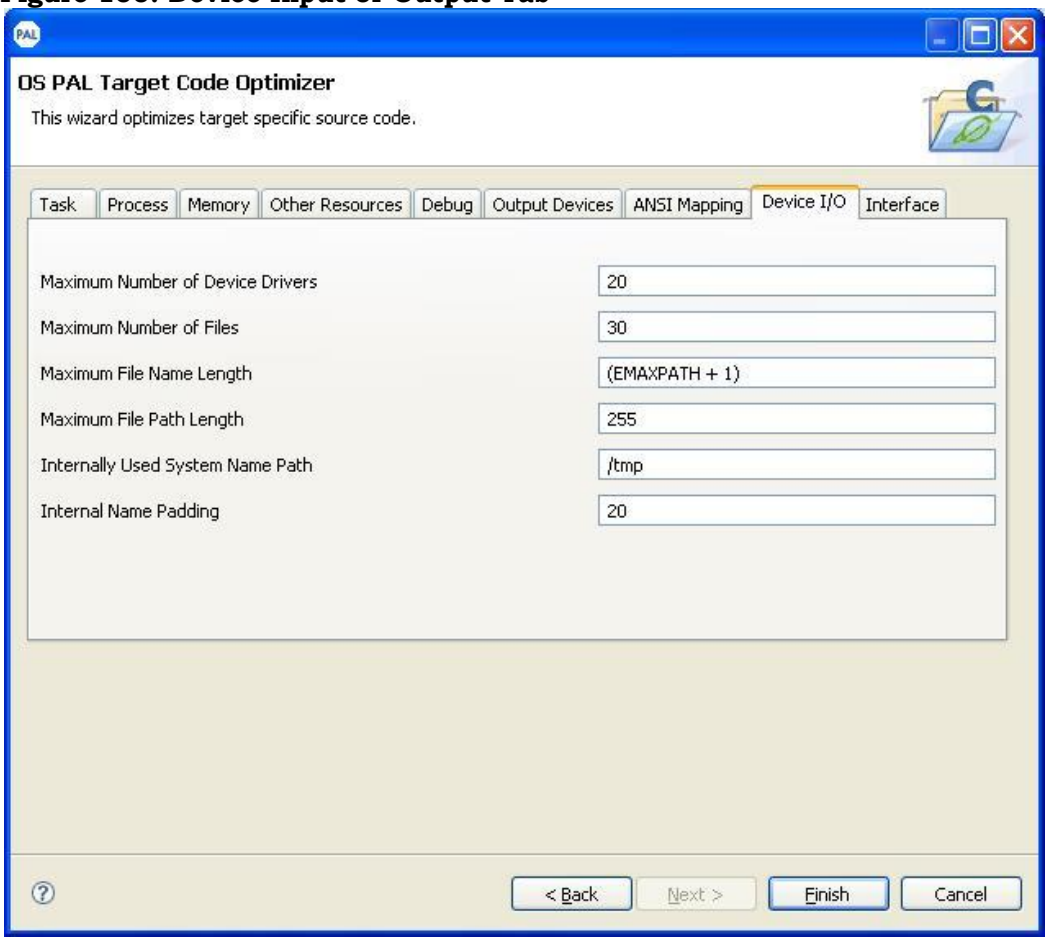


The field descriptions on ANSI Mapping tab are as follows:

| Field | Description | Your Action |
|---|---|---|
| Map ANSI Memory API | Specifies you want to map ANSImalloc() and free() to Cross-OS Interface equivalent functions. | To map ANSI to Cross-OS Interface equivalent functions, select the check box.<br>**Note**: By default this feature is enabled. |
| Map ANSI I/O API | Specifies if you want to map ANSI device I/O functions like open(), close(), read(), write, ioctl(), etc. to Cross-OS Interface equivalent functions. | To map ANSI I/O functions to Cross-OS Interface equivalent functions, select the check box.<br>**Note**: By default this feature is disabled. |
| MAP ANSI I/O Formatting API | Specifies if you want to map ANSIprintf() and sprintf() to Cross-OS Interface equivalent functions. | To map ANSI I/O formatting functions to Cross-OS Interface equivalent functions, select the check box.<br>**Note**: By default this feature is disabled. |

17. On **Device I/O** configuration tab, configure the options to your specifications as shown in Figure 100.

**Figure 100: Device Input or Output Tab**



The field descriptions on Device I/O tab are as follows:

| Field | Description | Your Action |
|---|---|---|
| Maximum Number of Device Drivers | Specifies the maximum number of drivers allowed in the Cross-OS Interface driver table structure.<br>**Note**: This excludes the native drivers the system, since they do not use the Cross-OS Interface driver table structure. | Enter the maximum number of device drivers.<br>**Note**: Default value is 20. |
| Maximum Number of Files | Specifies the maximum number of files that can be opened simultaneously using the Cross-OS Interface file control block structure.<br>**Note**: One control block is used when theCross-OS Interface driver is opened. These settings do not impact the OS setting for max number of files. | Enter the maximum number of files that can be opened simultaneously.<br>**Note**: Default value is 30. |
| Maximum File Name Length | Specifies the maximum length of the file name. | Enter the maximum number of files that can be opened simultaneously.<br>**Note**: Default value is Maximum File Path Length +1. |
| Maximum File Path Length | Specifies the maximum length of the directory path name including the file name for Cross-OS Interface use excluding the null char termination. | Enter the maximum length of the file path.<br>**Note**: Default value is 255.<br>This setting does not impact the OS setting for the max path/file name. |
| Internally Used System Name Path | Specifies the temporary directory of the file path. | Enter the temporary directory of the file path.<br>**Note**: Default value is /tmp. |
| Internal Name Padding | Specifies the padding for the internal name. | Enter the padding for the internal name.<br>**Note**: Default value is 20. |

18. If your project uses pSOSInterface, in **Interface** configuration tab, assign the number of unsigned arrays used to store the task's data as shown in

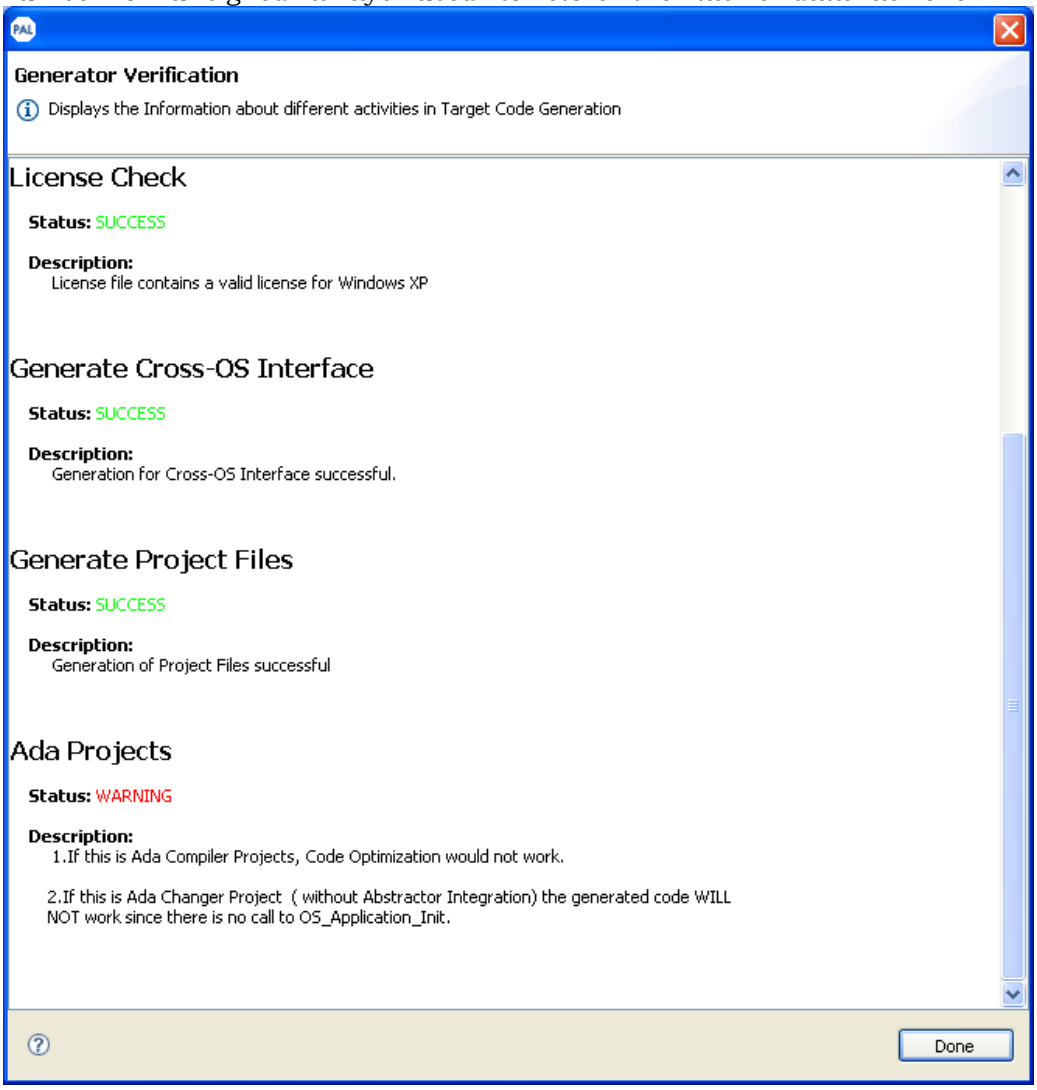
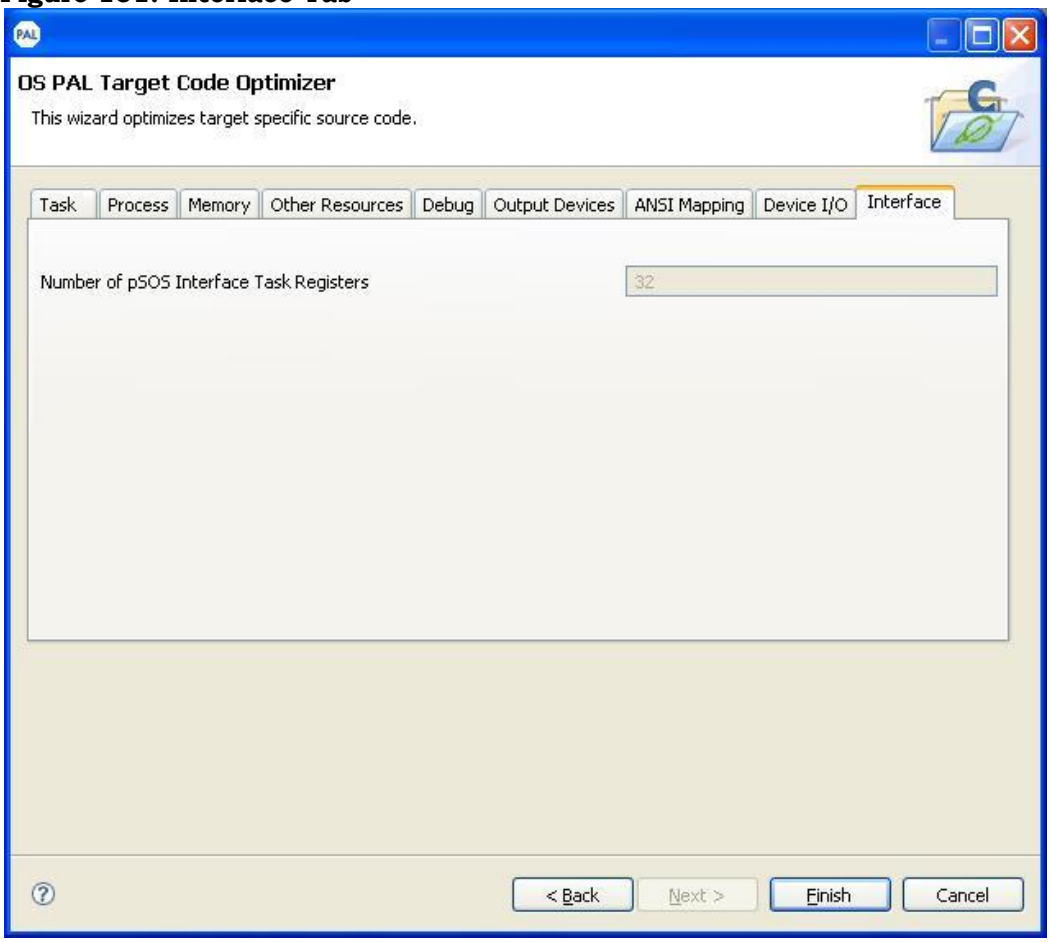



Figure 101.

**Figure 101: Interface Tab**

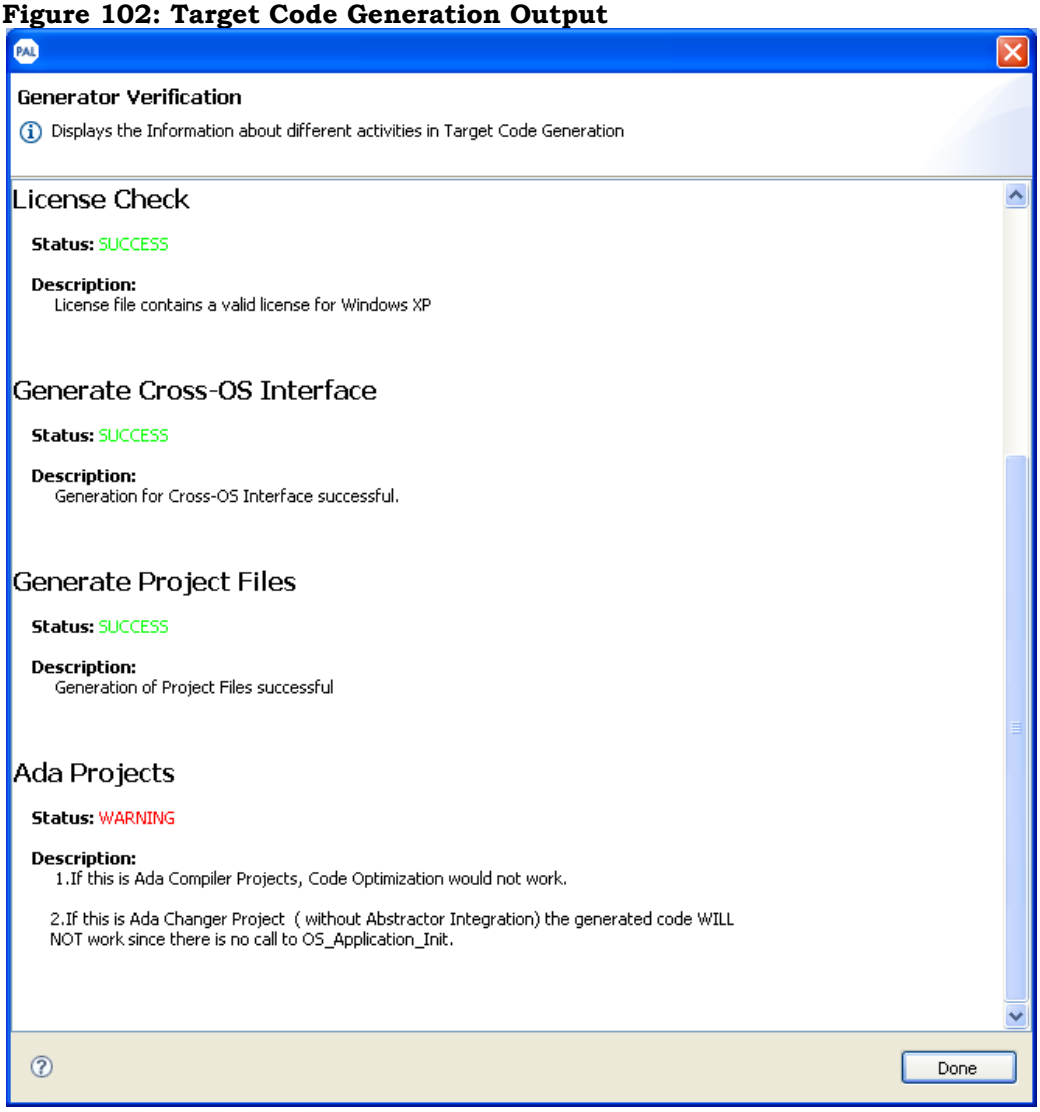


The field descriptions on Interface tab are as follows:

| Field | Description | Your Action |
|---|---|---|
| Number of pSOSInterface Task Registers | Specifies the number of pSOSInterface Task Registers. | Enter the number of pSOSInterface task registers.<br>**Note**: Default value is 32. |

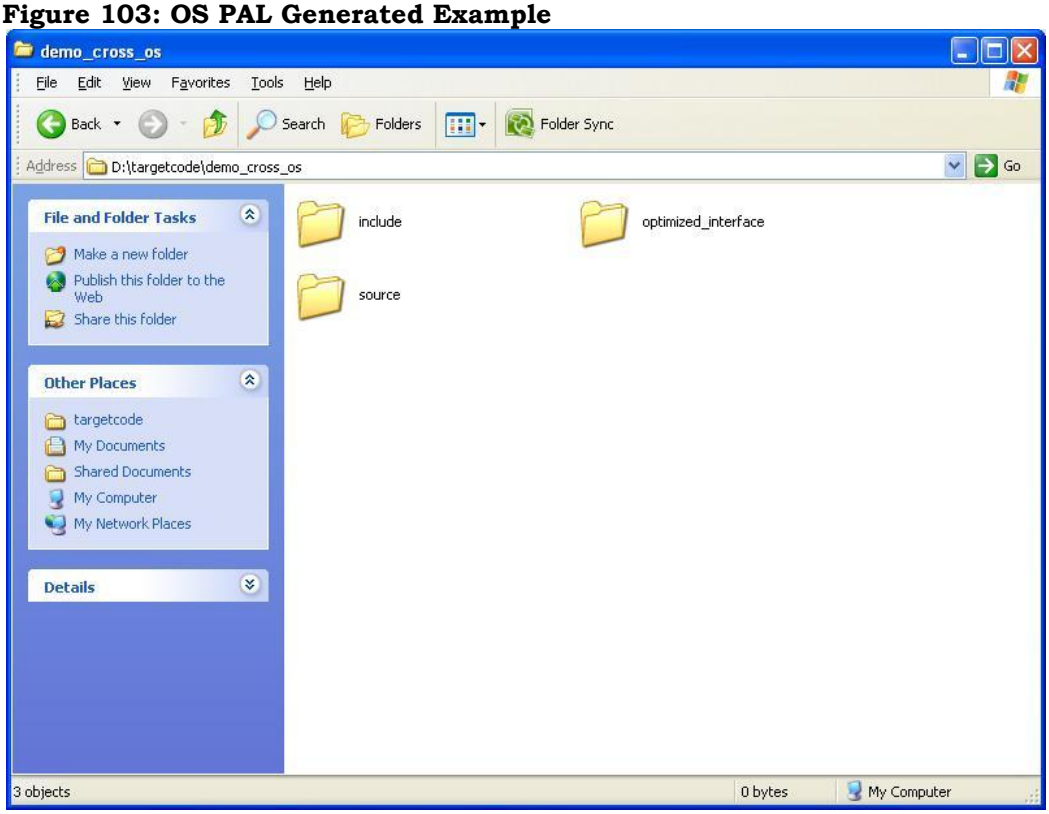
19. Click **Finish**. The target code will be generated into the destination path you defined in step 5 as shown in Figure 102.
    **NOTE**: If it is not able to generate the target code, the system will throw up an error.

**Figure 102: Target Code Generation Output**



20. You can view the OS PAL generated optimized code in Figure 103.

**Figure 103: OS PAL Generated Example**



# Generating Project Files for your Target

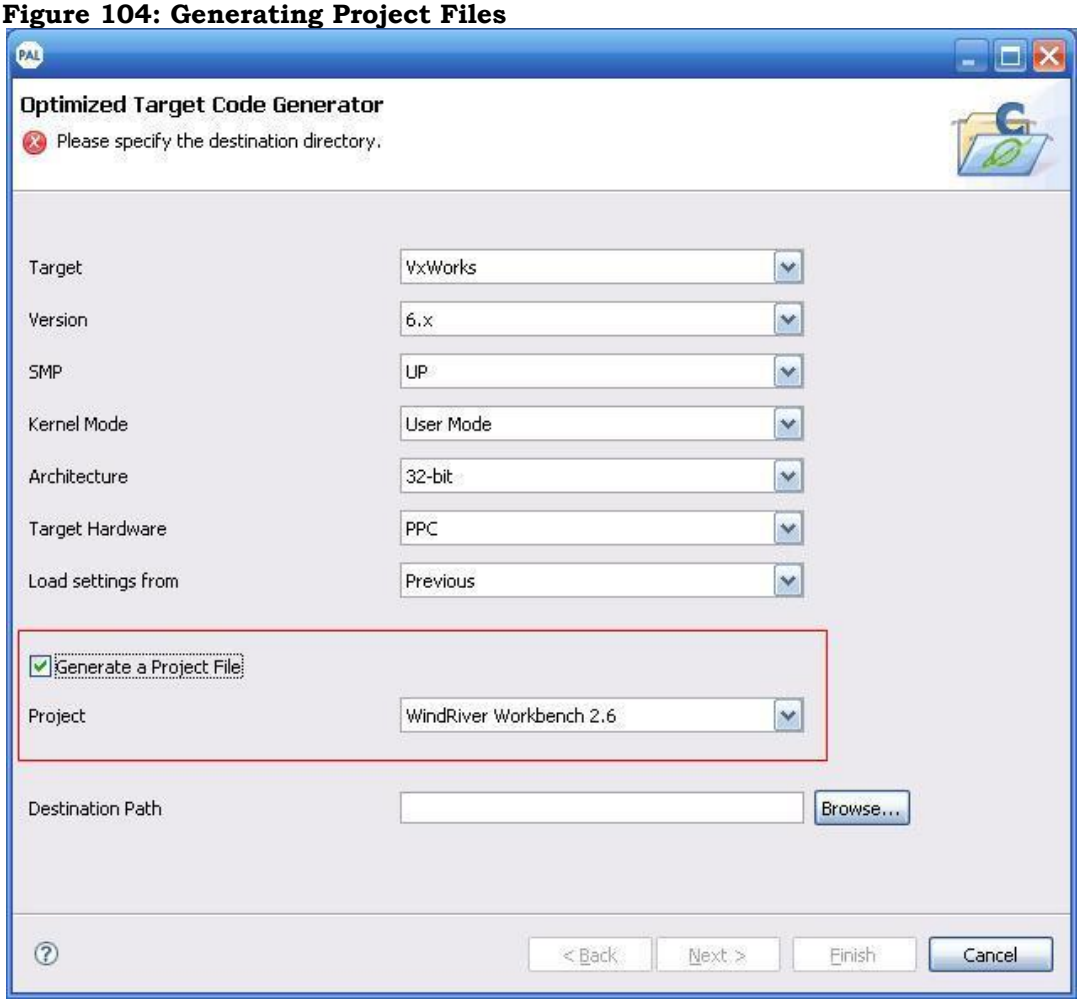
**NOTE**: This feature requires a target license. Click http://mapusoft.com/contact/ to send a request to receive licenses and documentation.

OS PAL provides the ability to generate project files for project files for the following targets:

- Wind River's Workbench
- QNX's Momentics
- Sun Microsystem's Sun Studio
- Visual Studio.NET 2005
- Micro Soft's Visual Studio 2006
- Micro Soft's Visual Studio .Net 2008
- Eclipse's CDT
- and makefiles

AfterGenerating Optimized Target Code,select the check box next to **Generate a Project File** and choose your IDE as shown inFigure 104.

**Figure 104: Generating Project Files**



# Inserting Application Code to Run only on Target OS Environment

The user configuration is done by setting up the appropriate value to the pre-processor defines found in the cross_os_usr.h.

**NOTE**: Make sure the OS Abstractor libraries are re-compiled and newly built whenever configuration changes are made to the cross_os_usr.h when you build your application. In order to re-build the library, you would actually require the full-source code product version (not the evaluation version) of OS Abstractor.

Applications can use a different output device as standard output by modifying the appropriate functions defines in os_target_usr.h along with modifying os_setup_serial_port.cmodule if they choose to use the format Input/Output calls provided by the OS Abstractor.

You can add some application code or target specific things such as memory allocations such as Heap Size and Shared memory which are specific to target environments.

## Target OS Selection

Based on the OS you want the application to be built, set the following pre-processor definition in your project setting or make files:

| Flag and Purpose | Available Options |
|---|---|
| **OS_TARGET**<br>To select the target operating system. | The value of the OS_TARGET should be for the Cross-OS Interface product that you have purchased. For Example, if you have purchased the license for :<br>**OS_NUCLEUS** – Nucleus PLUS® from ATI<br>**OS_THREADX** – ThreadX® from Express Logic<br>**OS_VXWORKS** – VxWorks® from Wind River Systems<br>**OS_ECOS** – eCOS standards from Red Hat<br>**OS_MQX** - Precise/MQX® from ARC International<br>**OS_UITRON** – micro-ITRON standard based OS<br>**OS_LINUX** - Open-source/commercial Linux® distributions<br>**OS_WINDOWS** – Windows 2000, Windows XP®, Windows CE, Windows Vista from Microsoft. If you need to use the Cross-OS Interface both under Windows and Windows CE platforms, then you will need to purchase additional target license.<br>**OS_TKERNEL** – Japanese T-Kernel® standards based OS<br>**OS_LYNXOS** - LynxOS® from LynuxWorks<br>**OS_QNX** – QNX operating system from QNX<br>**OS_LYNXOS** – LynxOS from LynuxWorks<br>**OS_SOLARIS** – Solaris from SUN Microsystems<br>**OS_ANDROID** – Mobile Operating System running on Linux Kernel<br>**OS_NETBSD** – UNIX like Operating System<br>**OS_UCOS** – UCOS® from Micrium<br>For example, if you want to develop for ThreadX, you will define this flag as follows:<br>OS_TARGET = OS_THREADX<br>PROPRIETARY OS: If you are doing your own porting of Cross-OS Interface to your proprietary OS, you could add your own |

| Flag and Purpose | Available Options |
|---|---|
| | define for your OS and include the appropriate OS interface files within os_target.h file. MapuSoft can also add custom support and validate the OS Abstraction solution for your proprietary OS platform |

# Running OS PAL Generated Code on your Target

**NOTE**: This feature requires a license and documentation.

Click http://mapusoft.com/contact/ to send a request to receive licenses and documentation.

After Generating Optimized Target Code for your target OS using the OS PAL Optimized Target Code Generator,

1. Using a cross-compiler, compile, link, and download the OS PAL generated code to your target.
2. Port low level drivers and hardware interrupt code as required (refer to Cross-OS Interface I/O and device driver APIs sections in the reference manual).
3. Resolve any run time errors.

# Building Cross-OS Interface Library

Before using Cross-OS Interface, make sure the OS and tools are configured correctly for your target. To ensure this, compile, link and execute a native sample demo application that is provided by the OS vendor on your target. Refer to the OS vendor provided documentation on how to compile, link, download, and debug the demo applications for your specific target and toolset. After this step, you are ready to use the Cross-OS Interface library to develop your applications.

## Building Cross-OS Interface Demo Application

The demo application is located at the \mapusoft\demo_cross_os directory location. From this location, you will find the make files or project files at the appropriate specific/<OS>/<tool>/<target> directory. For instance, if you need the demo application to be built for Nucleus PLUS OS using visual studio 6 tools and for x86 target, then the make file location will be at specific\nucleus\visual_studio_6\x86 directory.

# VxWorks Interface

The Nucleus Interface library contains the following modules:

| Module | Description |
|---|---|
| vxworks_interface.h | This header file is required in all of the VxWorksPLUS source modules. This header file provides the translation layer between the VxWorks defines, APIs and parameters to OS Abstraction |

The VxWorks Interface demo contains the following modules:

| Module | Description |
|---|---|
| demo.c | Contains a sample demo application |

## Building VxWorks Interface

Before building the VxWorksInterface library and/or application, ensure that the flag INCLUDE_OS_VxWorks is set to OS_TRUE in the cross_os_usr.h configuration file.

## Building VxWorks Interface Library

The VxWorksInterface library is located at \mapusoft\ vxworks_interface directory. From this location, you will find the make files or project files at the appropriate specific/<OS>/<tool>/<target> directory. For instance, if you need the demo application to be built for VxWorks OS using Eclipse tools and for x86 target, then the make file location will be at specific\vxworks\<OS>\x86\eclipse directory.

## Building VxWorks Interface Demo Application

The demo application is located at the \mapusoft\ demo_vxworks directory location. From this location, you will find the make files or project files at the appropriate specific/<OS>/<tools>/<target> directory. For instance, if you need the demo application to be built for VxWorks OS using Eclipse tools and for x86 target, then the make file location will be at specific\vxworks_interface\<OS>\x86\eclipse directory.

# POSIX Interface

The POSIX Interface library contains the following modules:

| Module | Description |
|---|---|
| posix_interface.h | This header file is required in all of the POSIX source modules. This header file provides the translation layer between the POSIX defines, APIs and parameters to OS Abstraction |

The POSIX Interface demo contains the following modules:

| Module | Description |
|---|---|
| demo.c | Contains a sample demo application |

## Building POSIX Interface

Before building the POSIX Interface library and/or application, ensure that the flags INCLUDE_OS_POSIX is set to OS_TRUE in the cross_os_usr.h configuration file.

## Building POSIX Interface Library

The POSIX Interface library is located at \mapusoft\posix_interface directory. From this location, you will find the make files or project files at the appropriate specific/<OS>/<tool>/<target> directory. For instance, if you need the demo application to be built for POSIX OS using Eclipse tools and for x86 target, then the make file location will be at specific\posix\<OS>\x86\elipse directory.

## Building POSIX Interface Demo Application

The demo application is located at the \mapusoft\demo_posix directory location. From this location, you will find the make files or project files at the appropriate specific/<OS>/<tools>/<target> directory. For instance, if you need the demo application to be built for POSIX OS using Eclipse tools and for x86 target, then the make file location will be at specific\posix\<OS>\x86\eclipse directory. We need to have the Cross-OS Interface Library. It has to be included in all the Interface demos.

After every demo application, include/link in the POSIX Interface library.

# Nucleus Interface

The Nucleus Interface library contains the following modules:

| Module | Description |
|---|---|
| nucleus_interface.h | This header file is required in all of the Nucleus PLUS source modules. This header file provides the translation layer between the Nucleus PLUS defines, APIs and parameters to OS Abstraction |

The Nucleus Interface demo contains the following modules:

| Module | Description |
|---|---|
| demo.c | Contains a sample demo application |

## Building Nucleus Interface

Before building the Nucleus Interface library and/or application, ensure that the flag INCLUDE_OS_Nucleus is set to OS_TRUE in the cross_os_usr.h configuration file.

## Building Nucleus Interface Library

The pSOS Interface library is located at \mapusoft\nucleus_interface directory. From this location, you will find the make files or project files at the appropriate specific/<OS>/<tool>/<target> directory. For instance, if you need the demo application to be built for Nucleus OS using Eclipse tools and for x86 target, then the make file location will be at specific\nucleus\<OS>\x86\eclipse directory.

## Building Nucleus Interface Demo Application

The demo application is located at the \mapusoft\ demo_nucleus directory location. From this location, you will find the make files or project files at the appropriate specific/<OS>/<tools>/<target> directory. For instance, if you need the demo application to be built for Nucleus OS using Eclipse tools and for x86 target, then the make file location will be at specific\nucleus\<OS>\x86\eclipse directory.

# pSOS Interface

The pSOS Interface library contains the following modules:

| Module | Description |
|---|---|
| psos_interface.h | This header file is required in all of the pSOS source modules. This header file provides the translation layer between the pSOS defines, APIs and parameters to OS Abstraction |

The pSOS Interface demo contains the following modules:

| Module | Description |
|---|---|
| demo.c | Contains a sample demo application |

## Building pSOS Interface

Before building the pSOS Interface library and/or application, ensure that the flag INCLUDE_OS_pSOS is set to OS_TRUE in the cross_os_usr.h configuration file.

## Building pSOS Interface Library

The pSOS Interface library is located at \mapusoft\ psos_interface directory. From this location, you will find the make files or project files at the appropriate specific/<OS>/<tool>/<target> directory. For instance, if you need the demo application to be built for pSOS OS using Eclipse tools and for x86 target, then the make file location will be at specific\psos_interface/<OS>\x86\eclipse directory.

## Building pSOS Interface Demo Application

The demo application is located at the \mapusoft\ demo_pSOS directory location. From this location, you will find the make files or project files at the appropriate specific/<OS>/<tools>/<target> directory. For instance, if you need the demo application to be built for pSOS OS using eclipse tools and for x86 target, then the make file location will be at specific\psos\<OS>\x86\eclipse directory.

# micro-ITRON Interface

The micro-ITRON Interface library contains the following modules:

| Module | Description |
|---|---|
| uitron_interface.h | This header file is required in all of the uITRON source modules. This header file provides the translation layer between the uITRON defines, APIs and parameters to OS Abstraction |

The Nucleus Interface demo contains the following modules:

| Module | Description |
|---|---|
| demo.c | Contains a sample demo application |

## Building micro-ITRON Interface

Before building the micro-ITRON Interface library and/or application, ensure that the flag INCLUDE_OS_UITRON is set to OS_TRUE in the cross_os_usr.h configuration file.

## Building micro-ITRON Interface Library

The micro-ITRON Interface library is located at \mapusoft\uitron_interfacedirectory. From this location, you will find the make files or project files at the appropriate specific/<OS>/<tool>/<target> directory. For instance, if you need the demo application

to be built for uITRON OS using Eclipse tools and for x86 target, then the make file location will be at specific\uitron\<OS>\x86\eclipse directory.

## Building micro-ITRON Interface Demo Application

The demo application is located at the \mapusoft\demo_uitrondirectory location. From this location, you will find the make files or project files at the appropriate specific/<OS>/<tools>/<target> directory. For instance, if you need the demo application to be built for micro-ITRON OS using Eclipse tools and for x86 target, then the make file location will be at specific\uitron\<OS>\x86\eclipse directory.

# Windows Interface

The Windows Interface library contains the following modules:

| Module | Description |
|---|---|
| windows_interface.h | This header file is required in all of the Windows source modules. This header file provides the translation layer between the Windows defines, APIs and parameters to OS Abstraction |

The Windows Interface demo contains the following modules:

| Module | Description |
|---|---|
| demo.c | Contains a sample demo application |

## Building WindowsInterface

Before building the WINDOWS Interface library and/or application, ensure that the flags INCLUDE_OS_WINDOWS and INCLUDE_OS_PROCESS are set to OS_TRUE in the cross_os_usr.h configuration file.

## Building WindowsInterface Library

The WINDOWS Interface library is located at \mapusoft\windows_interface directory. From this location, you will find the make files or project files at the appropriate specific/<OS>/<tool>/<target> directory. For instance, if you need the demo application to be built for WINDOWS OS using Eclipse tools and for x86 target, then the make file location will be at specific\windows\<OS>\x86\eclipse directory.

## Building WindowsInterface Demo Application

The demo application is located at the \mapusoft\demo_windows directory location. From this location, you will find the make files or project files at the appropriate specific/<OS>/<tools>/<target> directory. For instance, if you need the demo application to be built for Windows OS using Eclipse tools and for x86 target, then the make file location will be at specific\windows\<OS>\x86\eclipse directory. We need to have the Cross-OS Interface Library. It has to be included in all the Interface demos.

# Building Application with Multiple Interface Components

MapuSoft provides a feature to build application with multiple interfaces. For example; you can build an application with both Nucleus and VxWorks interfaces.

## Building Application with Multiple Interfaces

Before building the multiple Interface library and/or application, ensure that the corresponding flags are set to OS_TRUE in the cross_os_usr.h configuration file.

For instance; If you want to build an application with both Nucleus and VxWorks interfaces, set `INCLUDE_OS_NUCLEUS and INCLUDE_OS_VXWORKS` as OS_TRUE.

## Developing Applications with Multiple Interfaces

The steps for developing applications on host targets are described as follows:

1. Include os_target.h in all your application source files.
2. Set the appropriate compiler switches within the project build files to indicate the target OS and other target configurations.
3. Initialize the OS Abstractor library by calling OS_Application_Init() function. If you are also using POSIX Interface, then also use OS_Posix_Init() function call to initialize the POSIX component as well. For instance, to develop an application with both Nucleus and VxWorks application development, go to os_library_init.c and give your appropriate entry function in `NUCLEUS_ENTRY_FUNCTION`. Define the name of the Nucleus entry task. The default entry task is `NU_ROOT`. Give your appropriate entry function in `VXWORKS_ENTRY_FUNCTION`. You also have to give the appropriate stack size for your entry function in `VXWORKS_ENTRY_FUNCTION_STACK_SIZE`. The default stack size given by MapuSoft is `OS_MIN_STACK_SIZE`. In the main thread, call OS_Application_Wait_For_End() function to suspend the main thread and wait for application re-start or termination requests.
4. Compile and link your application using development tools provided by Mapusoft.
5. Download the complete application image to the target system and let it run.

Refer to the sample demo applications provided with OS Abstractor as a reference point to start your application. Please review the target processor and development tools documentation for additional information, including specific details on how to use the compiler, assembler, and linker.

# Chapter 8. OS PAL PROFILER

OS PAL provides the Profiler to collect performance data concerning your application and the platform. You can graphically view the data with charts and graphs to find bottlenecks system-wide or for a specific task. It enables you to generate API timing report and also do a comparison for two timing reports.

This chapter contains the following topics:

# About OS PAL Profiler

**NOTE**: This feature requires a license. Click http://mapusoft.com/downloads/ospal-evaluation/ to request an evaluation license.

The OS PAL Profiler is an add-on to the established OS PAL Eclipse based code migration and API optimization technology and is designed to enable data collection.

OS PAL Profiler offers the following:

- The data collected by the Profiler provides feedback concerning the utilization of MapuSoft's APIs in the project.
- The reports allow for performance impact analysis by detailing specific API execution time during a particular time period as well as the average and total API execution times.
- It enables you to collect data pertaining to the MapuSoft API's (Platform API profiling) and profiling user specific functions (Application Profiling).
- Users can analyze the data with the included OS PAL Profiler graphical viewer which offers area, bar, line, pie, and scatter charts, as shown in Figure 105.
- Profiler enables you to generate a Timing report to view the performance report for each API.
- OS PAL Profiler now enables you to generate Timing Comparison Report. This compares two different timing reports and compares the performance report for an API at different time and different values.

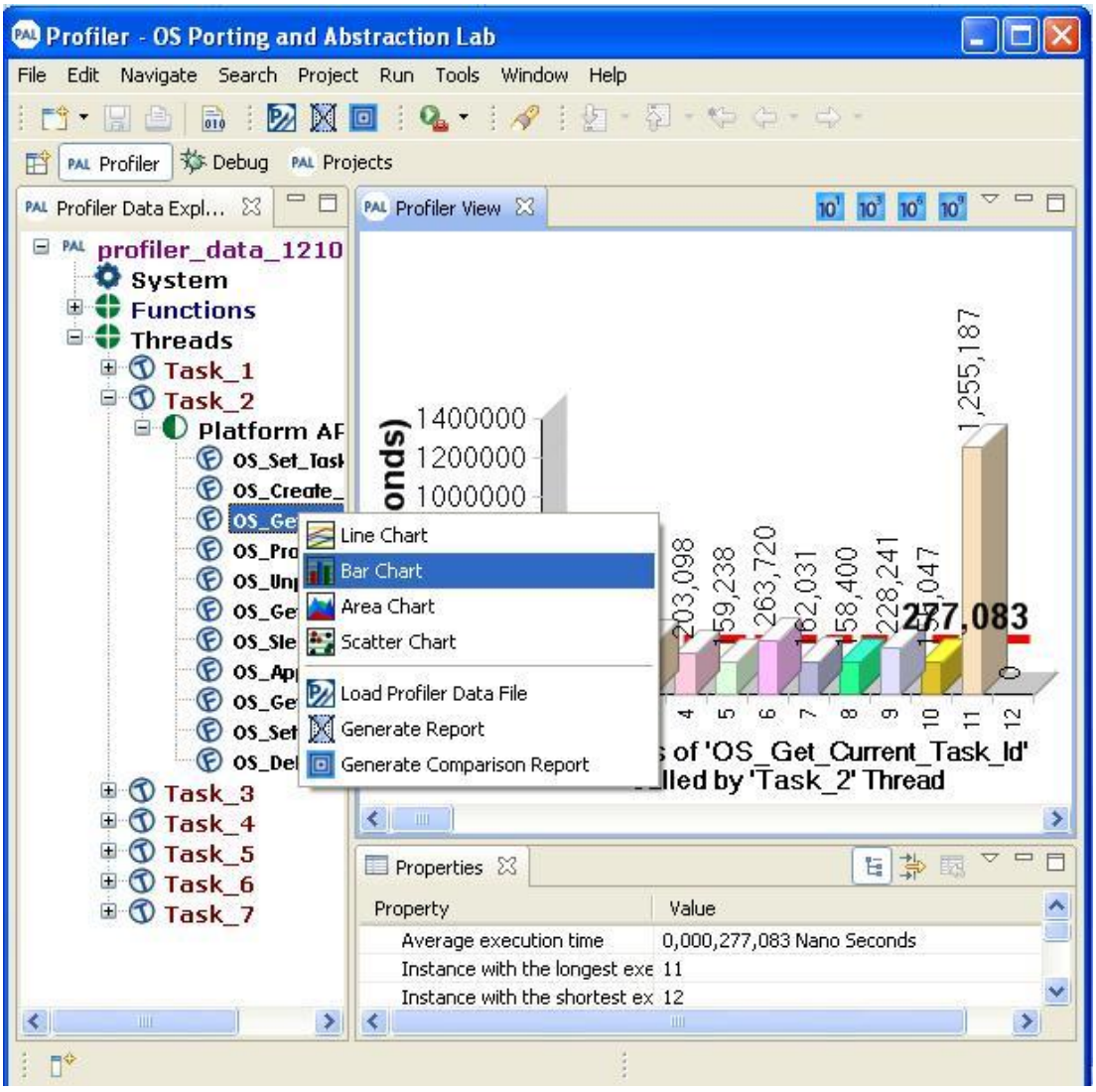
**NOTE 1:** In the current release, Profiler feature is not supported in ThreadX and Nucleus targets.

**NOTE 2**:The profiler feature does not generate profiler file XXX.PAL on Solaris target if you do code optimization for demo_cross_os with profiler ON. As a workaround, enter the following command at the prompt prior to running the demo:

prctl -n process.max-msg-qbytes -r -v 512KB -i process $$

The 512KB is the desired size of the queue and should be sufficient to run this example. If the number of messages is increased in cross_os_usr.h, then obviously this value will need to be adjusted.

**Figure 105: OS PAL Profiler**

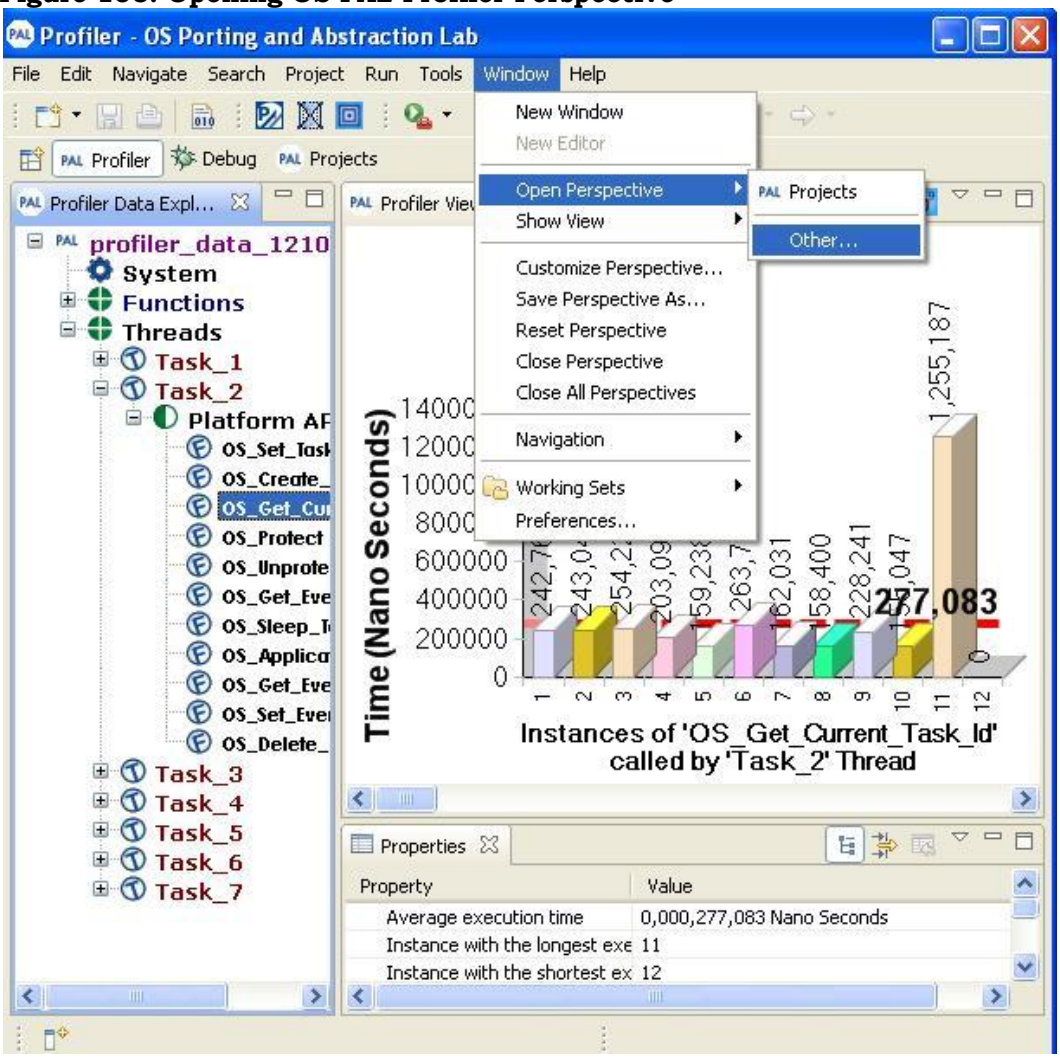


# Opening OS PAL Profiler Perspective

From OS PAL main menu, click **OS PAL Profiler** perspective button as highlighted.
Or,

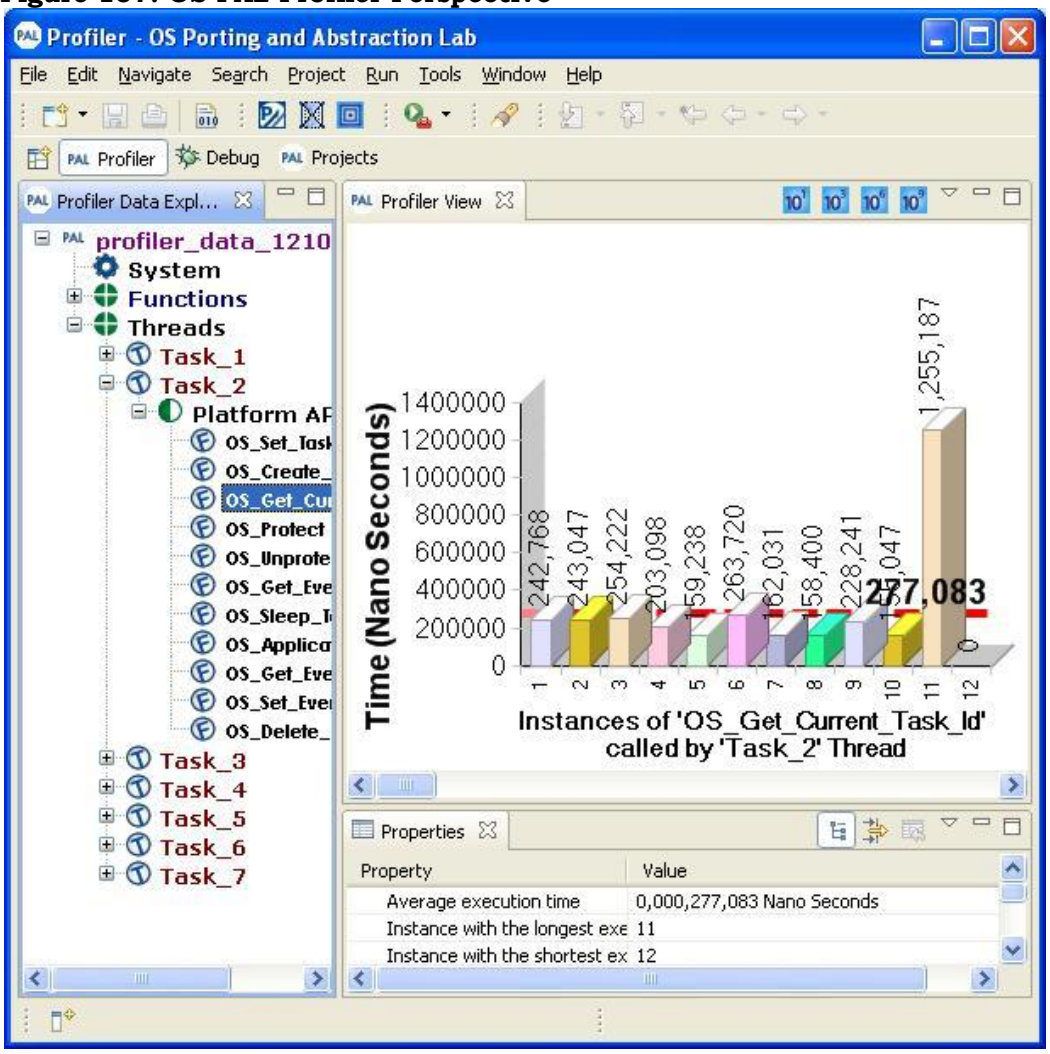
1. On OS PAL main menu, select **Window > Open Perspective > Profiler** as shown in Figure 106.

**Figure 106: Opening OS PAL Profiler Perspective**



You can view OS PAL Profiler Perspective as shown in Figure 107.

**Figure 107: OS PAL Profiler Perspective**



# Components on the Profiler Window

OS PAL Profiler window contains two panes. The left pane has three Profiler components listed and on the right pane, you can view the respective details and information in a graphical view.

The three main components of OS PAL Profiler are:

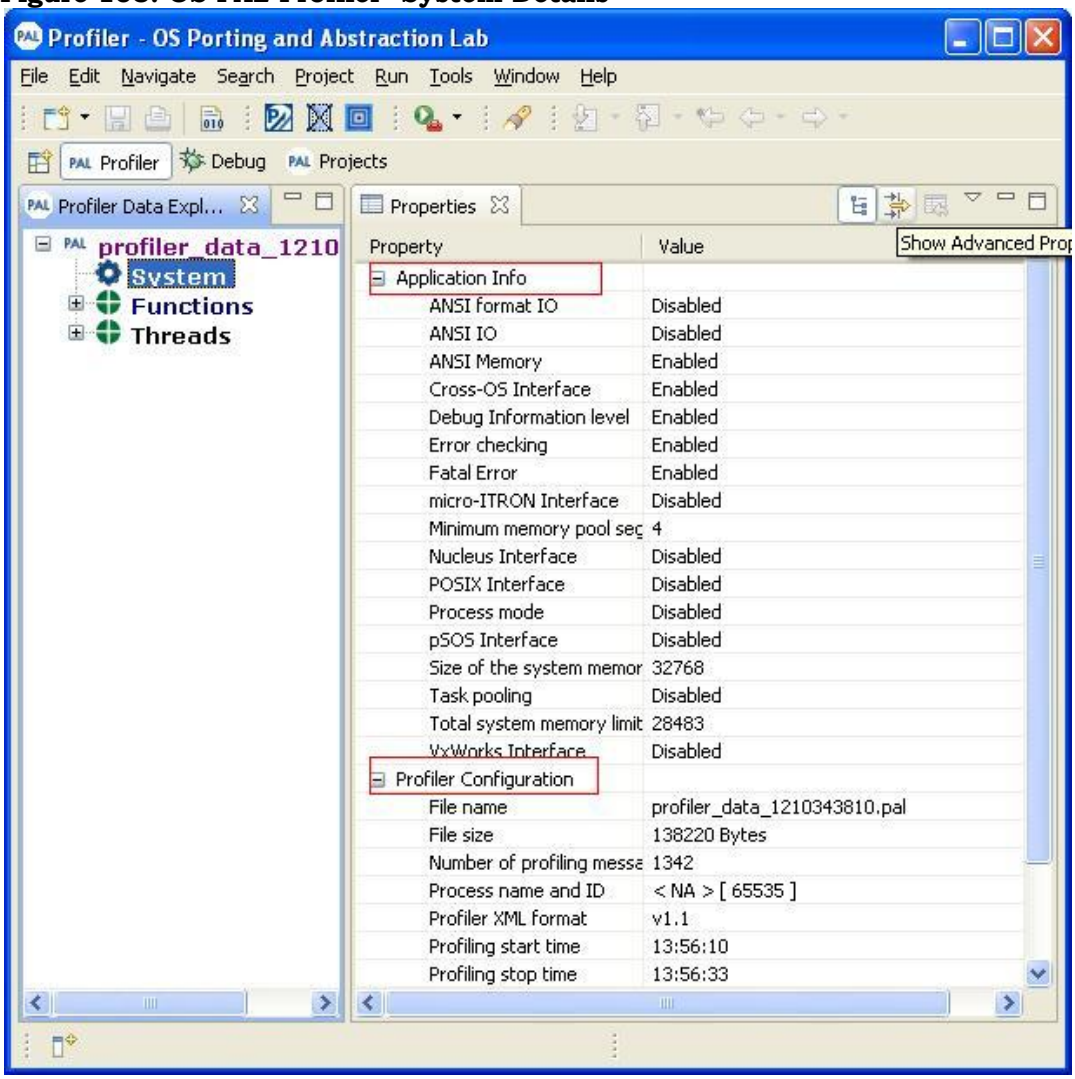
**Profiler_data file**–This is the generated profiler data file. You can view the performance report of each API. A profiler data file is saved as a .dat file. It has the following three components:

1. **System**–This displays the system details of your application as shown in Figure 108.

   If you selectSystem tab you have the following details which are displayed on the right pane as shown inFigure 108:

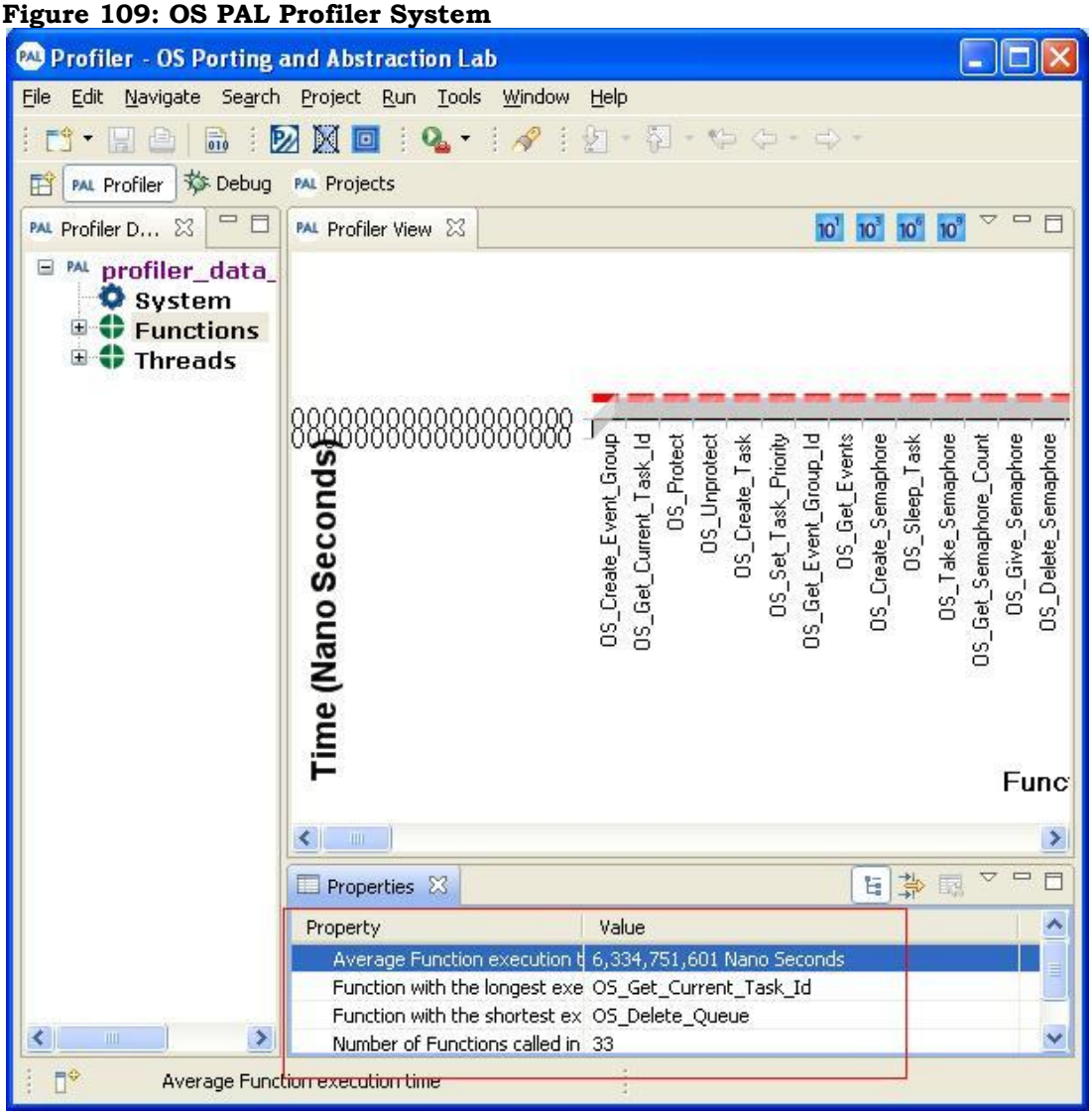
   - Application Info–Application property values
   - Profiler Configuration–Profiling Application values

**Figure 108: OS PAL Profiler- System Details**



2. **Functions**–This displays all the functions called in the application and the time taken to execute these functions as shown in Figure 109.

**Figure 109: OS PAL Profiler System**



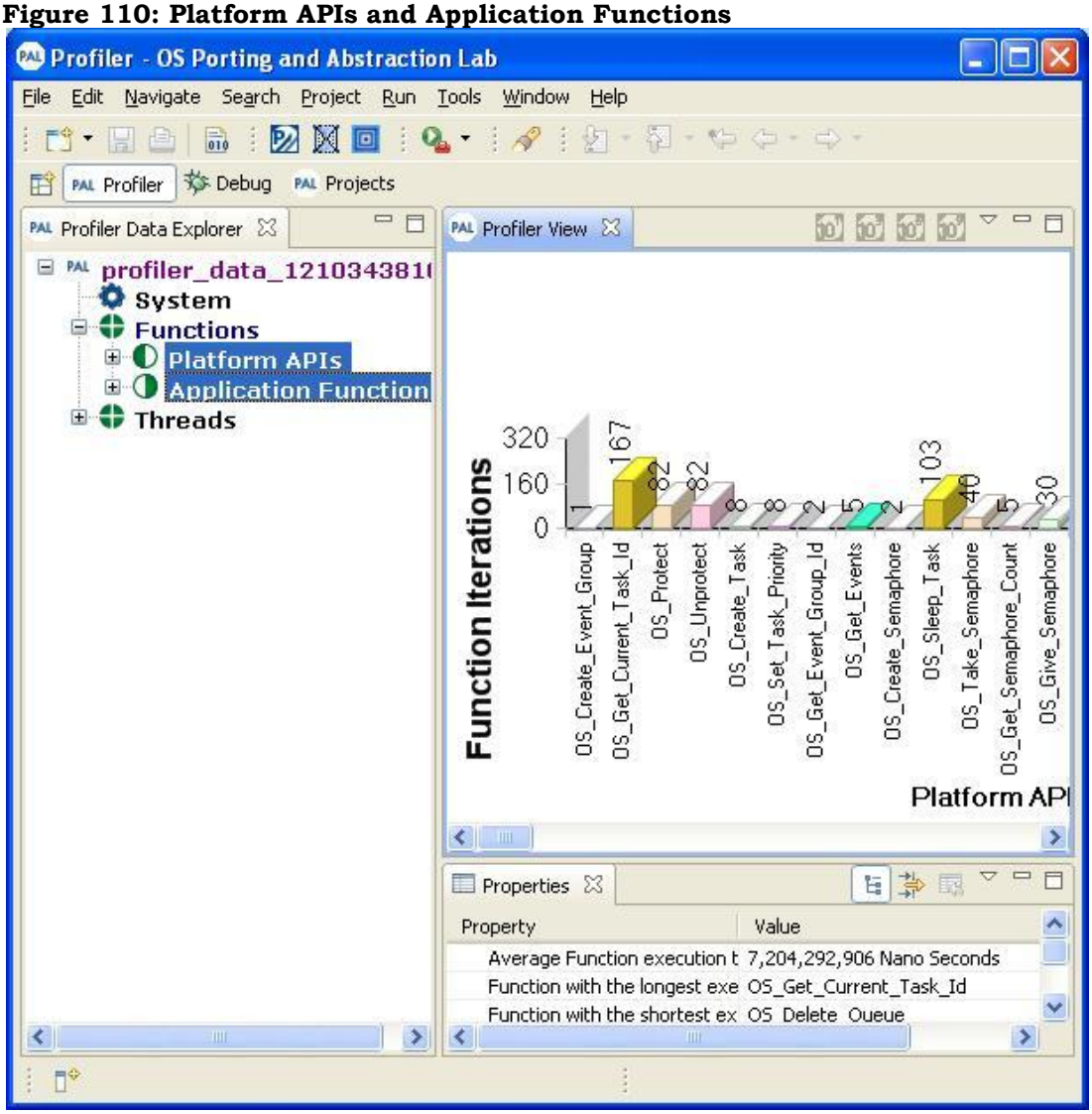

On the bottom of the window, as highlighted, the function properties are displayed such as:

- Average Function execution time
- Function with the longest execution time
- Function with the shortest execution time
- Number of Functions called in application

On the left pane, expand the **Functions** tab. It displays the following information as shown in Figure 110:

- Platform APIs
- Application Functions

**Figure 110: Platform APIs and Application Functions**




**Platform APIs**–These are all the Cross-OS Interface functions called in the application. On the x-axis, all the functions are displayed. On the y-axis, all functions iterations are displayed. On the bottom of the window the function properties are displayed such as:

- Average Function execution time
- Function with the longest execution time
- Function with the shortest execution time
- Number of Cross-OS Interface Functions called in application

If you expand the Platform APIs, you can view all the platform APIs called in the application as shown in Figure 110. On the bottom of the window, the function properties are displayed such as:

- Average execution time
- Instance with the longest execution time The *Task_1* in square brackets denote that these function properties belong to the Task 1 Thread.
- Instance with the shortest execution time
- Total number of times function was called

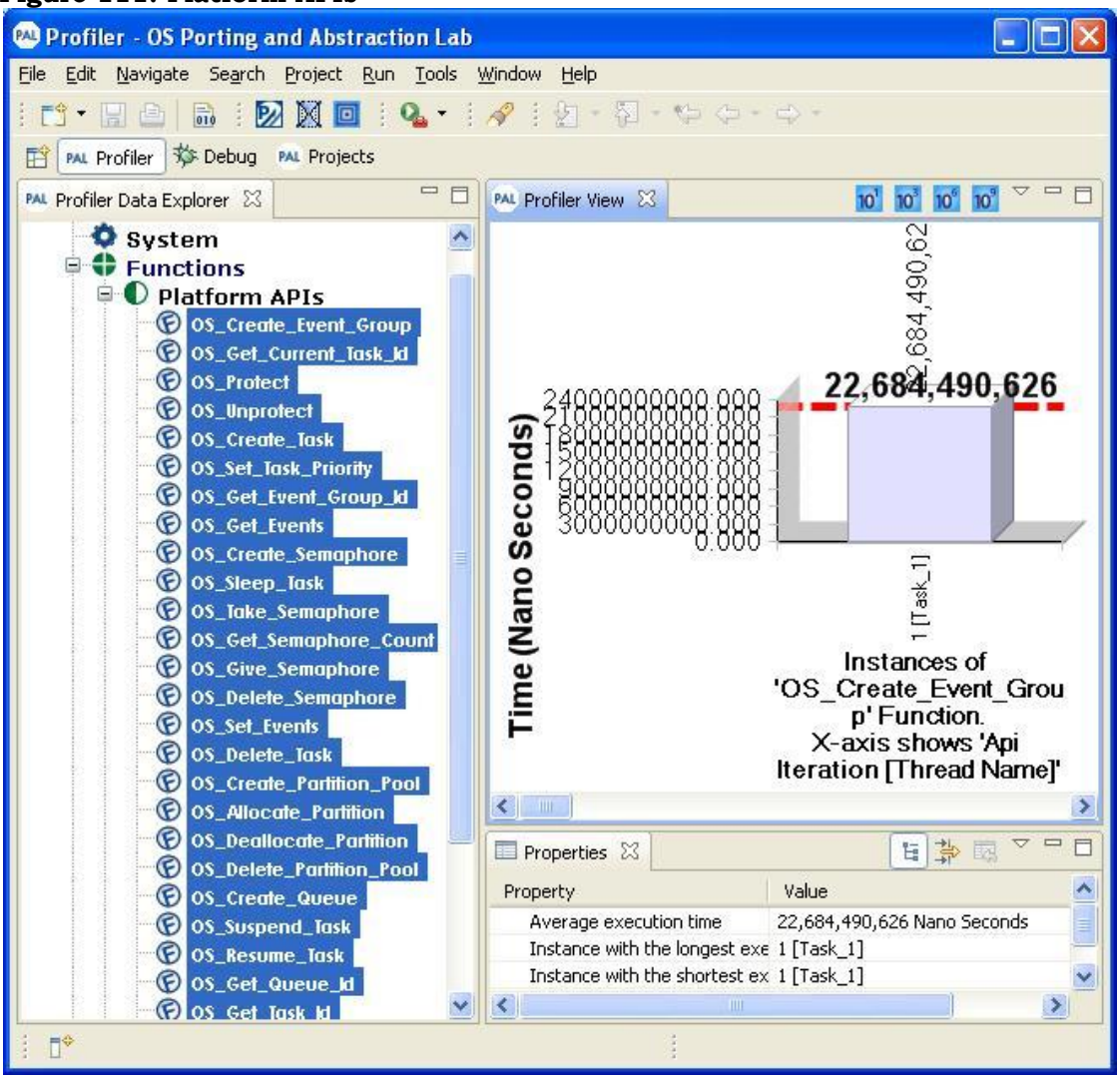
If you click on a Platform API, you can view the number of instances of the specific function on the x-axis and the time taken for each API on the y-axis.

**NOTE**: On top of the profiler view, you can view different measures of time such as:

- Seconds
- milli seconds
- micro seconds
- nano seconds as highlighted in the Figure 110.

This is used to capture the time taken for each instance of the function in different time measures. If you click on nano seconds, the time graph will be shown as Time (nano seconds) as shown in the Figure 111.

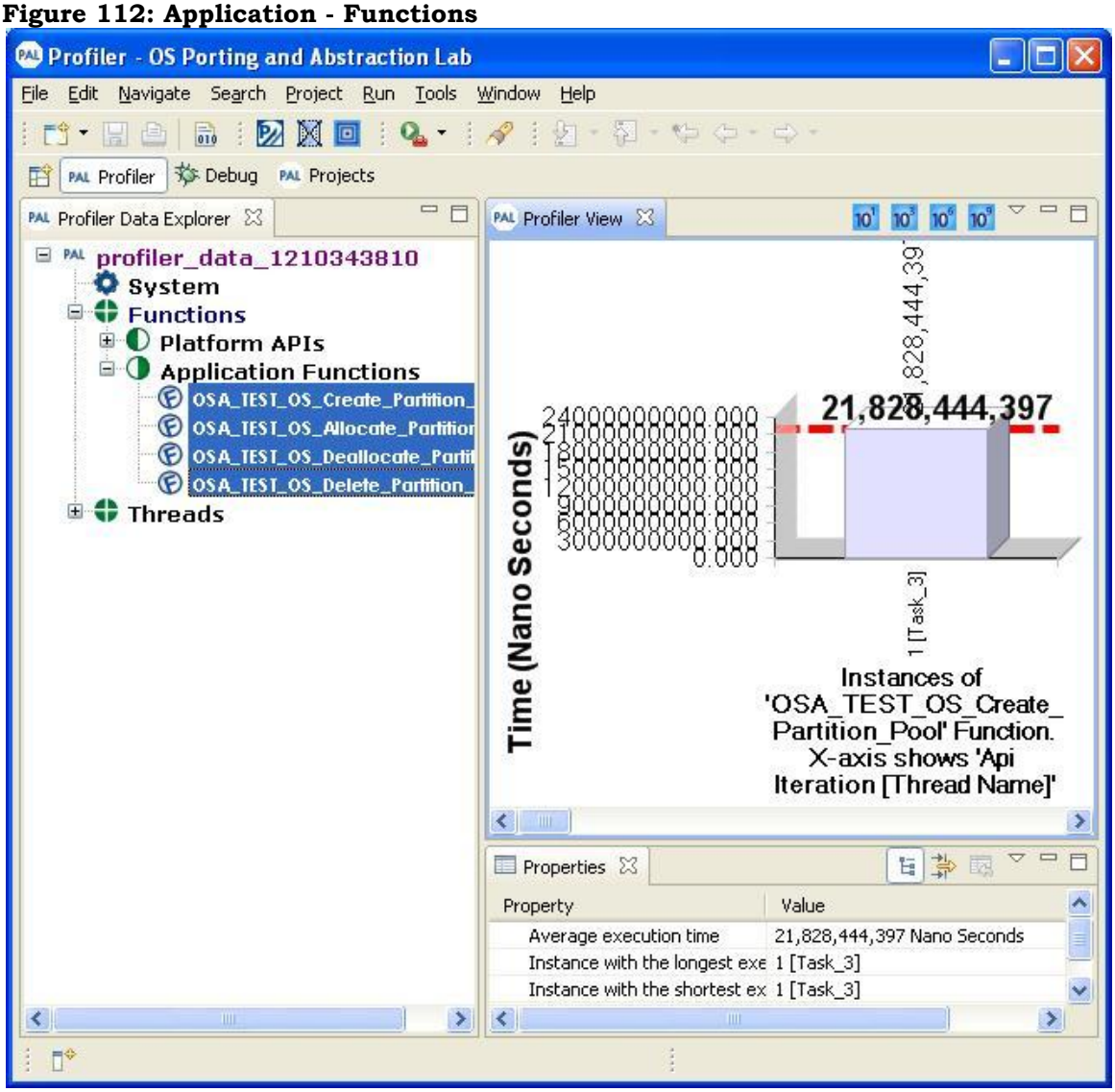
**Figure 111: Platform APIs**



**Application Functions**–These are all the user specific functions called in the application. On the x-axis, all the user specific functions are displayed. On the y-axis, all functions iterations are displayed. On the bottom of the window the function properties are displayed as shown in Figure 112 such as:

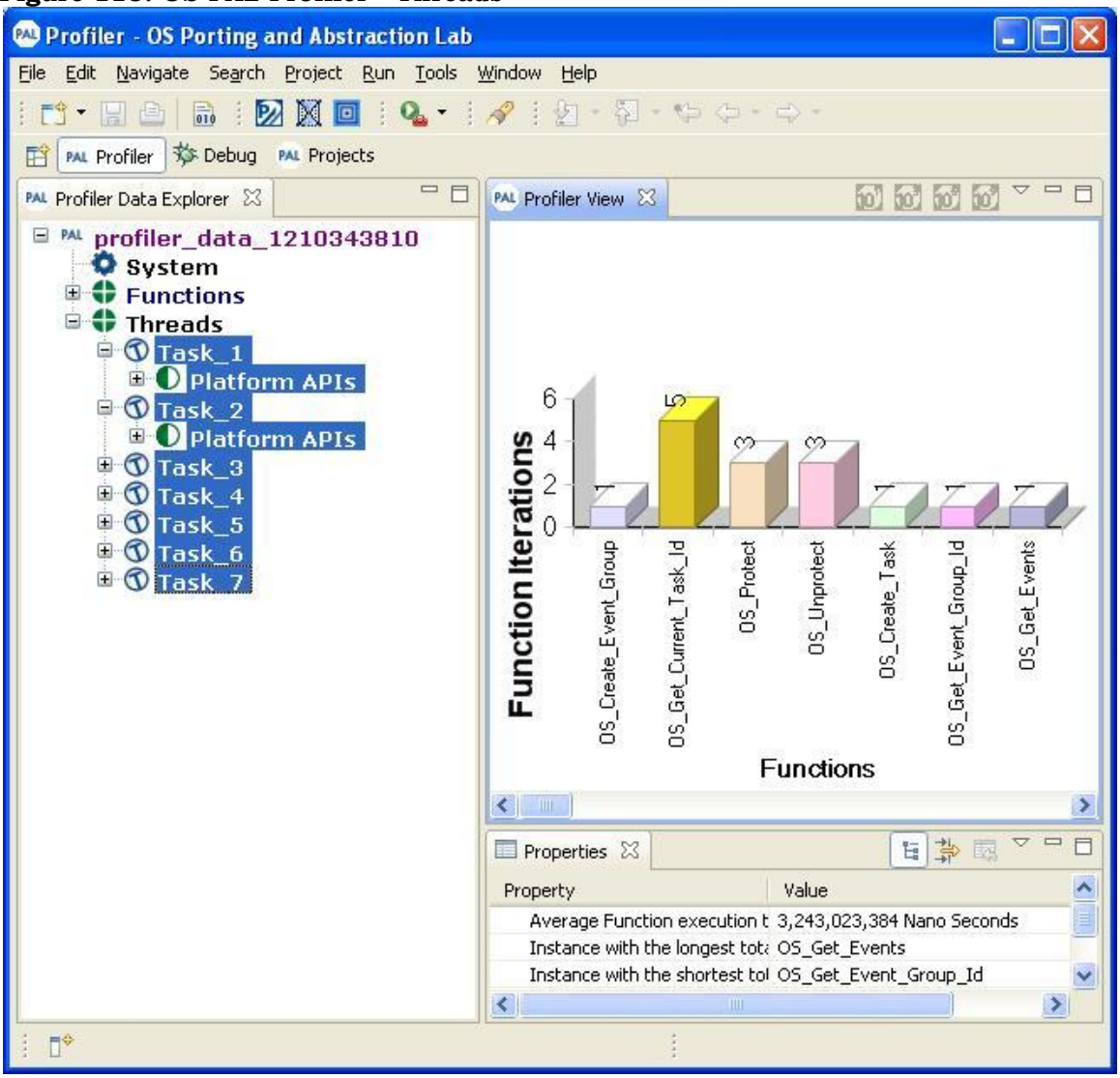
- Average Function execution time
- Function with the longest execution time
- Function with the shortest execution time
- Number of times Application Functions are called in application

**Figure 112: Application - Functions**



3. **Threads**–Threads are created to execute any function in an application. IN OS PAL Profiler you can view the Thread properties by expanding the Thread tab as shown in Figure 113. On the bottom of the window the thread properties are displayed as shown in Figure 113 such as:
   - Average Function execution time
   - Instance with the longest total execution time
   - Instance with the shortest total execution time
   - Number of Functions used by this thread
   - Thread ID

**Figure 113: OS PAL Profiler - Threads**




**Tasks**–These are functions called for each task. If you expand the Task tab, you have the following as already discussed under the Functions tab:

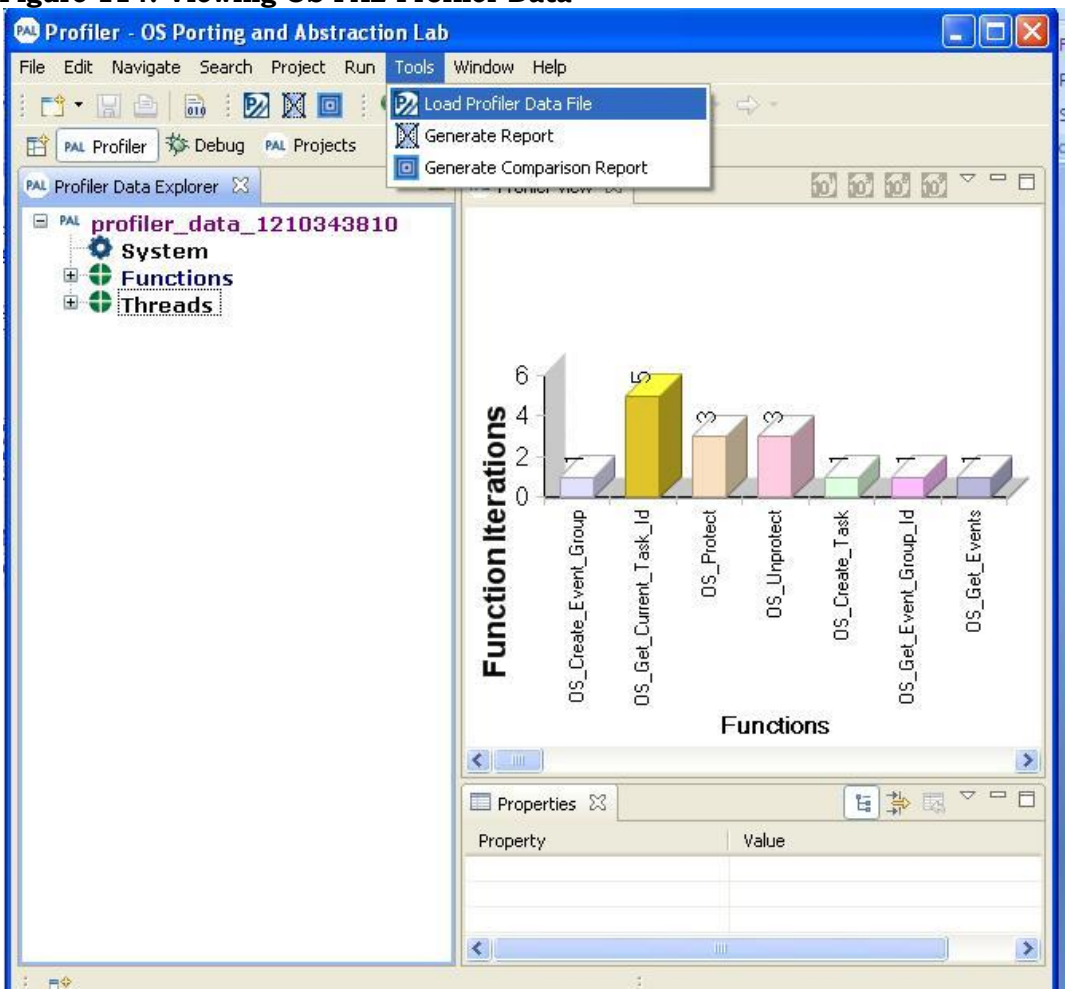
- Platform APIs
- Application Functions

# Viewing OS PAL Profiler Data

**NOTE**: This feature requires a license. Click http://mapusoft.com/downloads/ospal-evaluation/ to request an evaluation license.

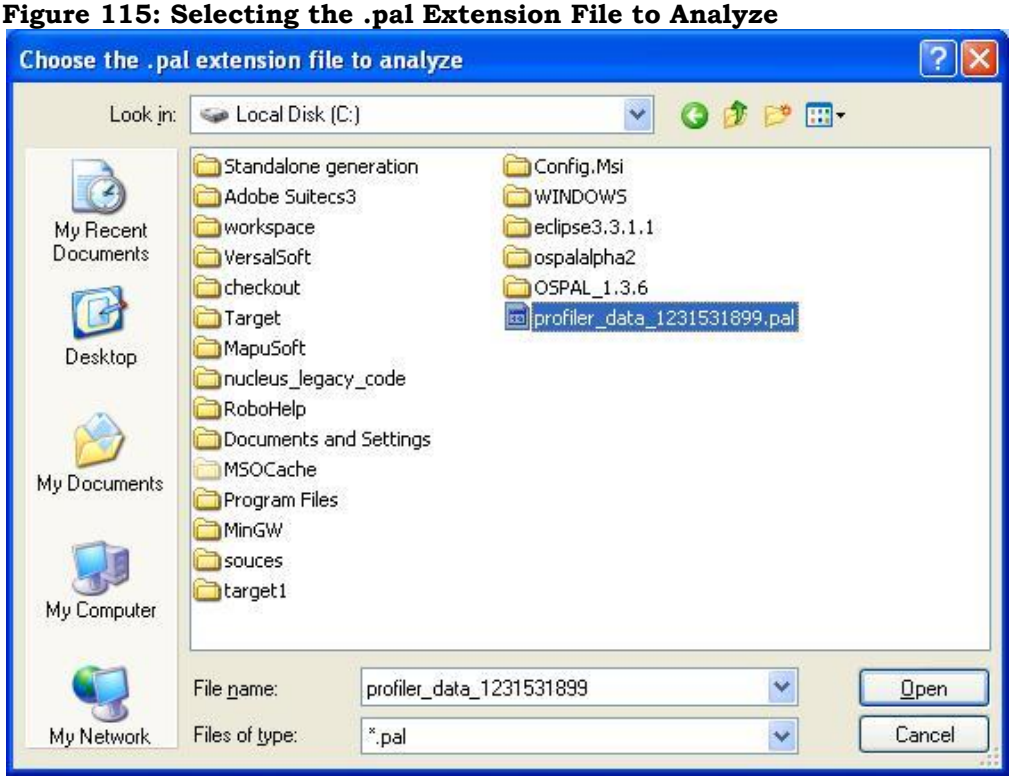
1. Open the OS PAL Profiler perspective.
2. From the OS PAL main menu, select **Tools >Load Profiler Data File** as shown in Figure 114.

**Figure 114: Viewing OS PAL Profiler Data**



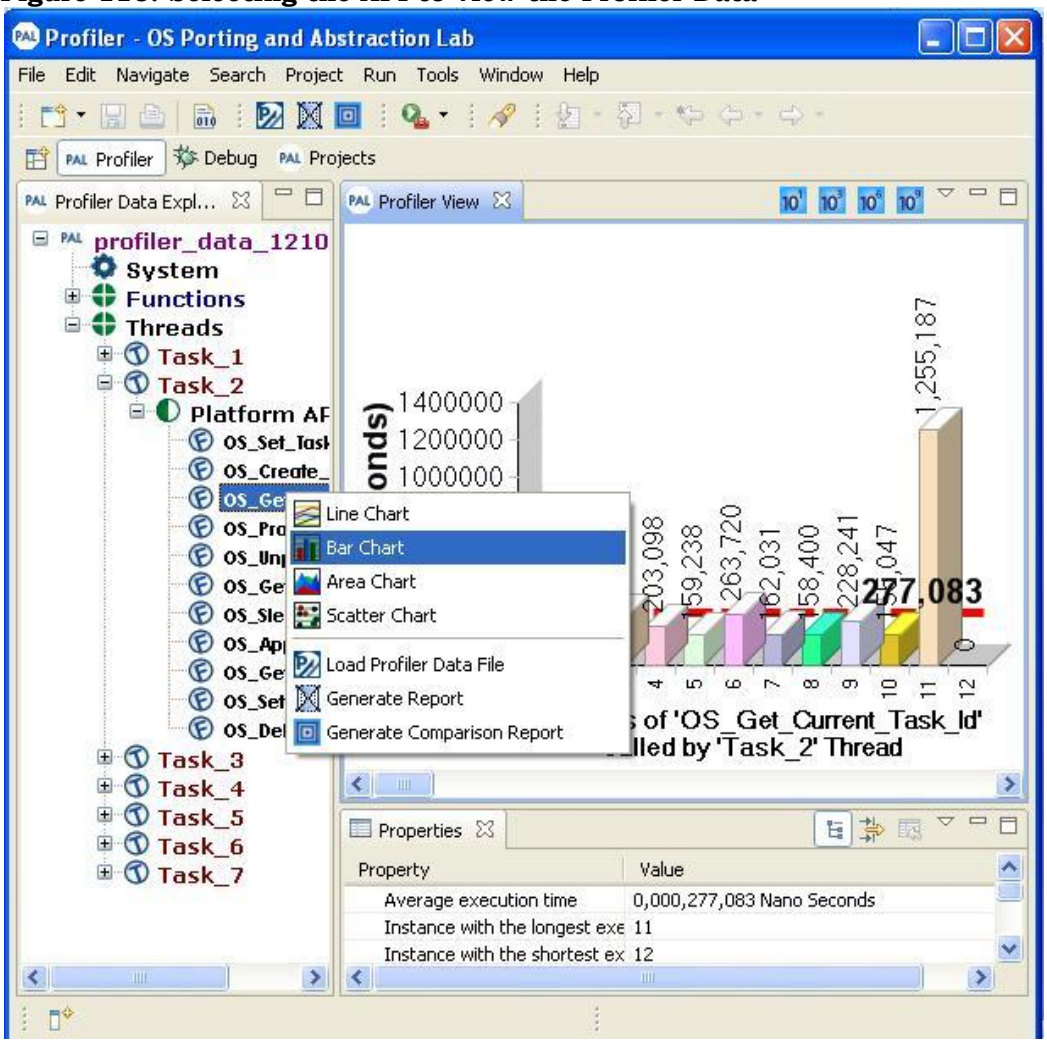
3. Browse to your saved profiler data file, and click **Open** as shown in Figure 115.

**Figure 115: Selecting the .pal Extension File to Analyze**



4. Select an API to view the data and right click on **Profiler Data Explorer** tab to view the different graph options as shown in Figure 116.
   **NOTE**: You can select an appropriate graphical viewer to view your profiler data. You can view the profiler data in a line chart, bar chart, area chart, or a scatter chart.

**Figure 116: Selecting the API to view the Profiler Data**



# Generating API Timing Report

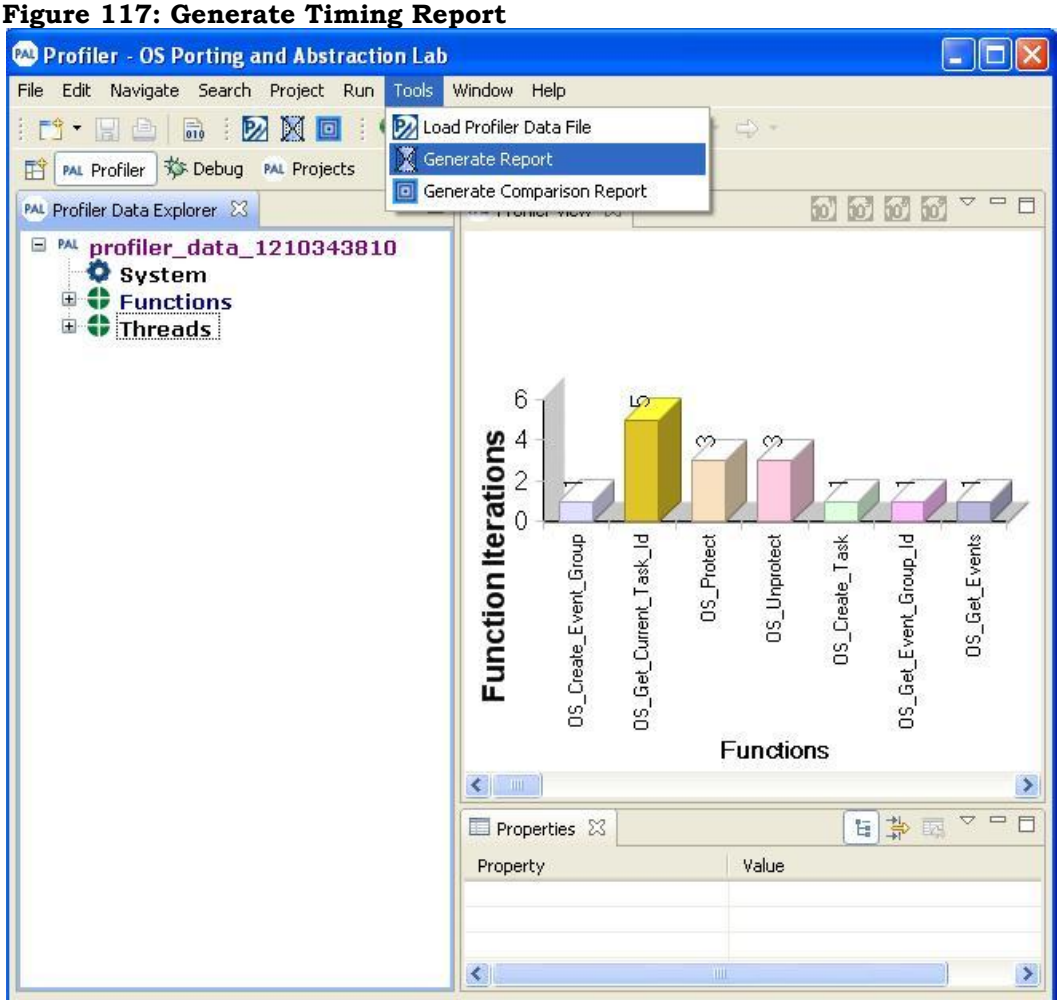
**NOTE**: This feature requires a license. Click http://mapusoft.com/downloads/ospal-evaluation/ to request an evaluation license.

OS PAL now provides you a new feature to view the performance report for each API.

To generate API Timing Report:

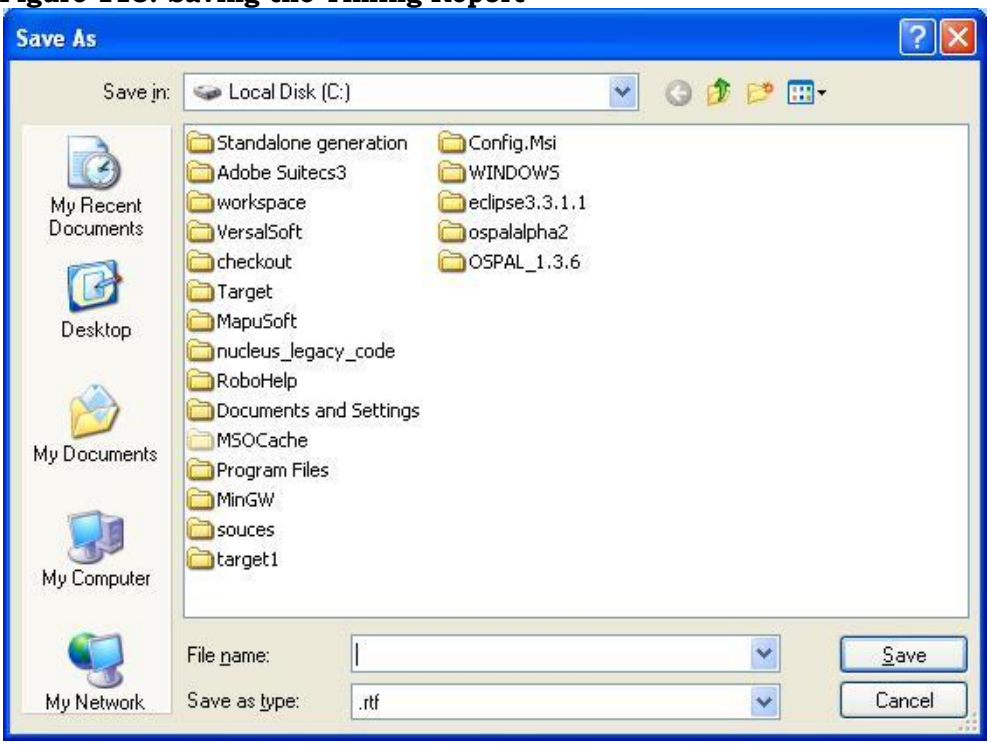
1. From the OS PAL main menu, go to Profiler perspective and select any Profiler Data on your left pane to generate the report.
2. Select **Tools > Generate Report**. You can also click on Generate Report button  on the OS PAL window as shown in Figure 117.

**Figure 117: Generate Timing Report**



3. A Save As window is displayed. Select the directory where you want to save the report and enter a file name for the report and click **Save** as shown in Figure 118.
   **NOTE**: In Windows Vista and Windows 7, you cannot generate the profiler report on c:\, if UAC is turned on. To turn off UAC, refer to the **Turning Off UAC**section on page 123. You can generate the Timing report to generate in any sub-folder inside C drive. For Ex:C:\pal_report.

**Figure 118: Saving the Timing Report**




4. Your Timing Report is successfully generated as an .rtf file.The Timing Report displays the following information:

1. **Timing Information—**The timing information gives a detailed description of the following:

   - Best Time Value – Specifies the minimum time taken to perform the action on each platform API
   - Worst Time Value – Specifies the maximum time taken to perform the action on each platform API
   - Average Time Value – Specifies the average time taken to perform the action on each platform API

2. **Application Information—**When you perform the application profiling on OS PAL, the report displays the following application property values:
   - Total system memory limit– Specifies the total system memory pool limit of the application.
   - Size of the system memory pool– Specifies the size of the system memory pool of the application.
   - Minimum memory pool segment size– Specifies the minimum size of the memory pool segment of the application.
   - VxWorks Interface–Specifies if you have enabled VxWorks Interface.
   - pSOS Interface– Specifies if you have enabled pSOS Interface.
   - POSIX Interface– Specifies if you have enabled POSIX Interface.
   - Cross-OS Interface– Specifies if you have enabled the Cross-OS Interface.
   - Process mode– Specifies if the Cross-OS Interface process feature is enabled or disabled.
   - Task pooling– Specifies if the Task pooling feature is enabled for this application.
   - ANSI Memory– Specifies if you want to map ANSImalloc() and free() to Cross-OS Interface equivalent functions.
   - ANSI format IO– Specifies if you want to map ANSIprintf() and sprintf() to Cross-OS Interface equivalent functions.
   - Debug Information level– Specifies if you want to enable the debug output.
   - Error checking– Specifies if you want to enable the error checking.
   - Fatal Error– Specifies if you want to enable the feature to ignore fatal errors.
3. **Profiler Configuration—**When you perform profiling on OS PAL APIs, the report displays the following profiling application values:
   - File name–Specifies the name of the .pal file generated by Cross-OS Interface
   - Project name–Specifies the name of your project
   - Target name–Specifies the target OS you have selected for profiling
   - File size–Specifies the size of the file to be profiled
   - Profiler XML format–Specifies the version of the XML used for profiling
   - Process name and ID–Specifies the process name and the ID
   - User Data–Specifies the information provided by the user
   - Profiling start time–Specifies the starting time of profiling
   - Profiling stop time–Specifies the end time of profiling
   - Total time profiled–Specifies the total time taken for profiling.
   - Number of profiling messages– Specifies the number of profiler messages.

# Generating Timing Comparison Report

**NOTE**: This feature requires a license. Click http://mapusoft.com/downloads/ospal-evaluation/ to request an evaluation license.

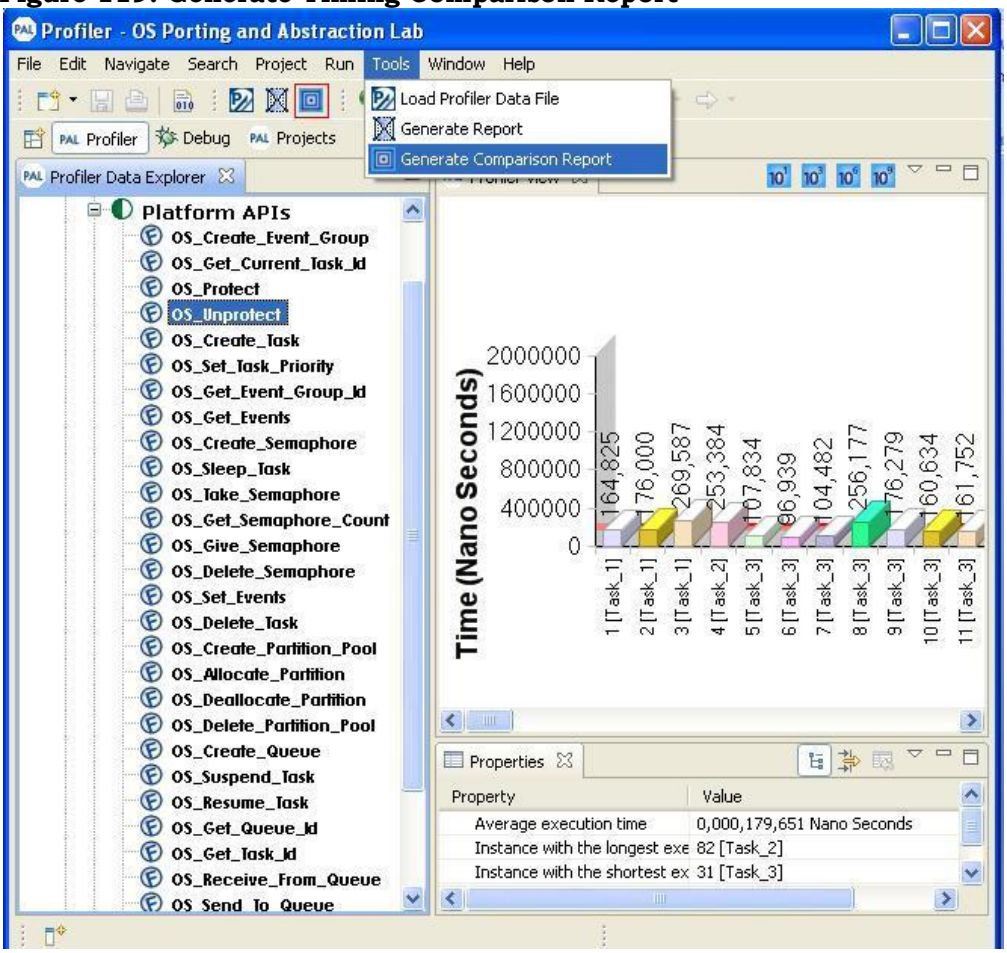
OS PAL now provides you a new feature to view the comparison of two different performance reports for the APIs. You can generate a timing comparison report only when the following preconditions are met:

- Both the PAL files must have the same project name.
- Both the PAL files must have a profiling time less than 5 minutes.

**To generate Timing Comparison Report**:

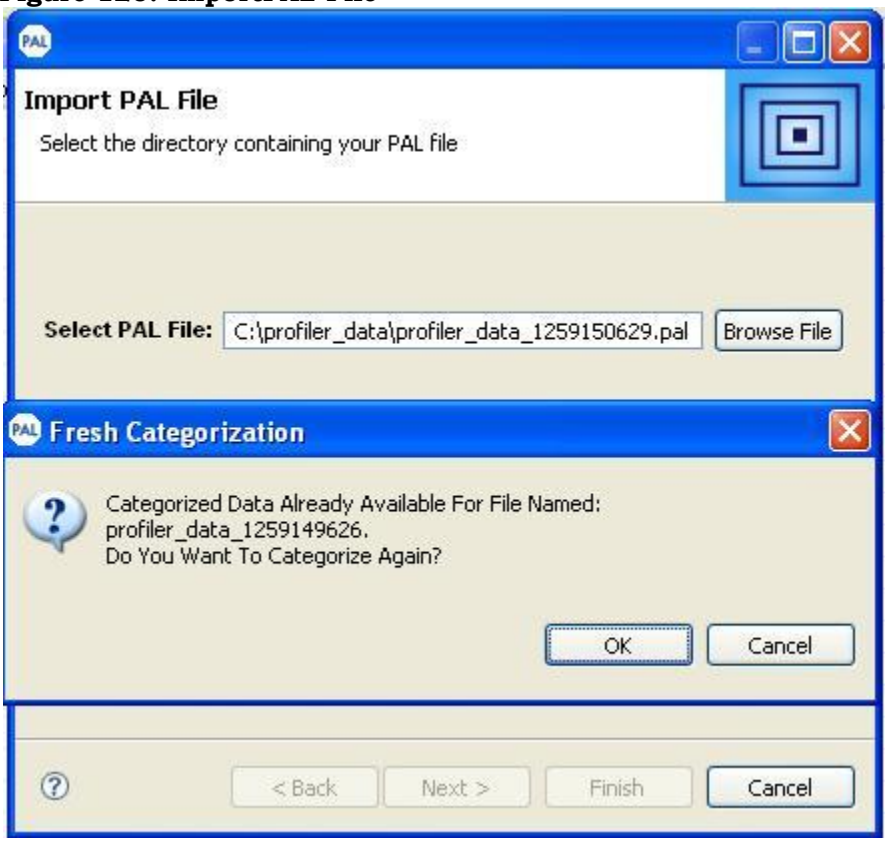
1. From the OS PAL main menu, go to Profiler perspective and select any Profiler Data on your left pane to generate the report.
2. Select **Tools > Generate Comparison Report**. You can also click on Generate Comparison Report button on the OS PAL window as shown inFigure 119.

**Figure 119: Generate Timing Comparison Report**



3. AnImport PAL file window is displayed. Select the PAL fileby clicking on the **Browse** button or entering the second PAL file path in the text box. A Profiler Categorization Dialog box is displayed. **NOTE**: If you comparing the two PAL files for the first time, click **OK**. If you are comparing the same two files for the second time, click **Cancel**. as shown in Figure 120.

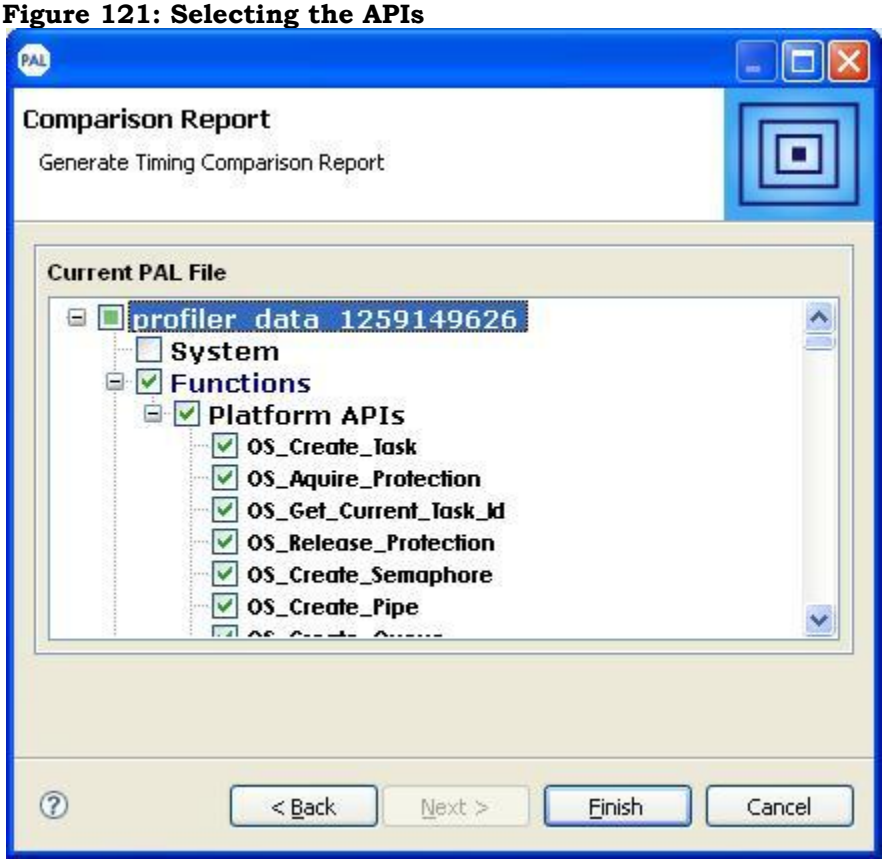
**Figure 120: ImportPAL File**



The field descriptions for importing the PAL file are described as follows:

| Field | Description | Your Action |
|---|---|---|
| Select PAL File | Specifies you to select the PAL file for which the Timing Comparison Report has to be generated. | To select the PAL file,click **Browse**, and select it from your system. |
| Report for | Specifies what you are comparing. | You can do any one of the following, and click **Next**:<br>• To compare only the differences, select the radio button before **Differential Data**. **Note**: By default, this feature is disabled.<br>• To compare all the data in the PAL files, select the radio button before **All Data**. **Note**: By default, this feature is disabled. |

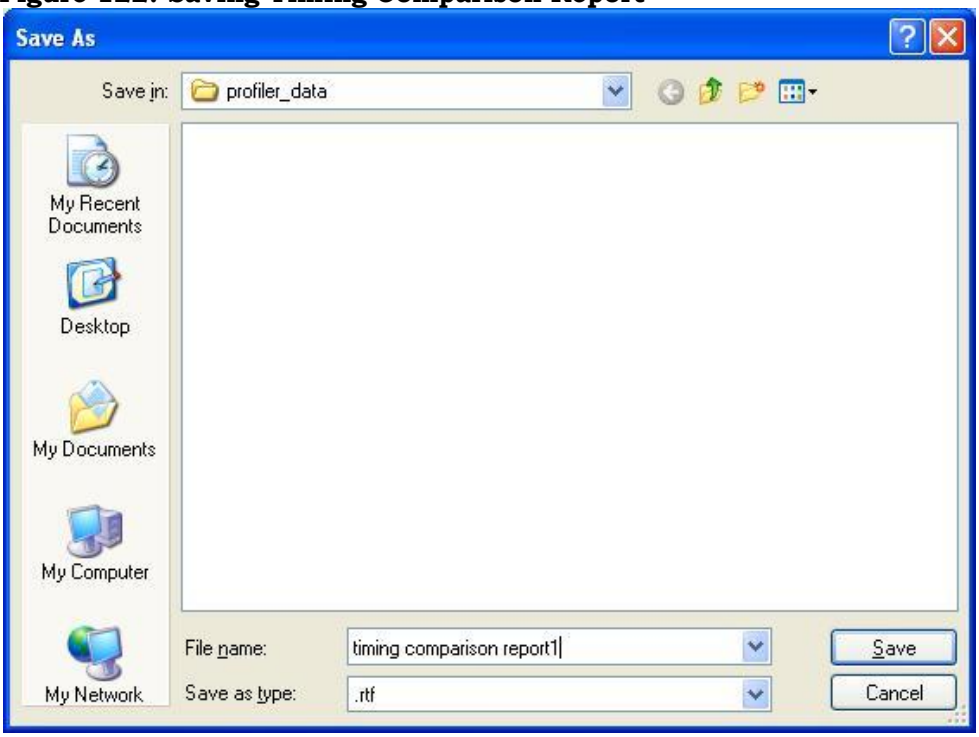
**Figure 121: Selecting the APIs**




4. Select the APIs you want to generate the Timing Comparison Report for and click **Finish**.

5. A Save As window is displayed. Select the directory where you want to save the report and enter a file name for the report and click **Save** as shown in Figure 122.

**Figure 122: Saving Timing Comparison Report**




Your Timing Comparison Report is successfully generated as an .rtf file.

# Chapter 9. Introduction to Ada Tool

## Ada Tool in OS PAL

**Note**: This feature requires a license. Click http://mapusoft.com/downloads/ospal-evaluationto request an evaluation license.

Ada tool integrated in OSPAL tool includes a compiler and a conversion tool. They are as follows:

- **Ada-PAL Compiler**–Ada-PAL Compiler functions within OS PAL to compile Ada code. The tool can also be integrated with various C/C++ cross-compilers to generate binaries for a wide variety of target platforms. This converts Ada sources in root directory to object or binary executables.
- **Ada-to-C/C++Changer**–Ada-C/C++ Changer allows developers to easily convert software written in Ada code to C/C++ utilizing OS PAL. The resultant C/C++ software can be integrated with the robust OS Abstractor® environment to support a wide variety of host and target OS platforms. The automatic conversion process eliminates the need for costly and tedious code rewrites, providing extensive resource savings. Ada-C/C++ Changer generates ANSI C output as well as certain C++ features while preserving Ada code's comments, files, structures and variable names to ease ongoing code maintenance.

# Using the Ada Root Directory

The Ada root directory contains all information needed to support the separate compilation requirements of Ada. The primary contents of the root directory are Ada source files, all object modules and info files created by the compiler are stored in the OSPAL Projects. Since no intermediate compilation form is saved, the Ada compiler performs a semantic analysis of the appropriate source files, as necessary, to handle any separate compilation requirements.

This source-based program library model simplifies the use of the Ada compiler and program builder:

- There are no compilation order requirements for compiling Ada source. As long as the Ada source for depended-upon units is available, it is not necessary to compile them first.
- There are no constraints on the user's approach to file organization or configuration management.
- There are no significant disk storage requirements for the program library beyond that required for the source and object modules.

## Root Directory

In Ada root directory, you inform the library of the location of the Ada source for the program. The source can be in one directory, in multiple directories, or in multiple program libraries. Once this information is provided, the Ada compiler and program builder can automatically locate source files containing the required units, as needed. The following subsections describe the program library, with details about how the Ada compiler and program builder use this program library.

Two tools are provided for maintaining the program library:

- The program library options tool, adaopts, can be used to display or modify the program library parameters. This creates ADA.LIB.
- The source registration tool, adareg, establishes which units are defined in which source files. A description of the use of these tools follows the description of the program library. This creates UNIT.MAP.

An Ada program library is based in a directory called the program library directory. All informationabout the program library and all generated files are kept in the program library directory (or inspecified subdirectories).The main contents of the program library are the source files and object modules. There isconsiderable flexibility with regard to the actual location of the source files. This will be evident in the examples that follow.

## Configuration with Multiple Source Directories

In a larger program, the source files composing the Ada program are often located in severaldirectories. To support this source configuration, the program library provides a source directory listwhich points to the directories containing the source.In this configuration, the source can be distributed in any convenient way among any number ofsource directories.

## Configuration with Linked Libraries

For more complex programming efforts, it may be desirable to partition the source code intosubsystems, each of which is maintained within a separate program library. To support this model,the Ada program library supports linking to other existing libraries.The user need not know the location of the source for a linked library, just its program librarydirectory. If the linked library is itself linked to another library, that library also needs to be added asa linked library for the current library. The source files and object modules of a linked library mayonly be referenced in a read-only fashion.

## Specifying the Configuration

The program library's configuration is determined by the values of program library parameters.The configuration described above is the default configuration created automatically by the firstinvocation of the Ada C/C++ Changer orAda-PAL compiler.The primary difference between the "multiple source directories" model and the "multiplelinked libraries" model is what happens when adabgen discovers that a source file needs tobe [re]compiled:

- If the source file is part of this program library, adabgen will recompile it.
- If the source file comes from a linked library, adabgen will refuse to recompile it, and will givean error message.

Therefore, the "multiple source directories" model is more convenient for most projects.

# Contents of the Ada Tools

## ADA.LIB and UNIT.MAP

An Ada tools contains two files—ADA.LIB and UNIT.MAP. These two files, which are located in the project are automatically created the first time the Ada compiler, is invoked. ADA.LIB contains information describing the configuration of the library. UNIT.MAP contains a unit-to-source mapping for use by the compiler and program builder. When the program library is created, if a UNIT.MAP file already exists in the current directory, it will be used for the new program library's UNIT.MAP.

## Source Files

The Ada source files in the program library include:

- All Ada source files in the program library directory
- All Ada source files in directories specified in the program library's source directory list.
- Any other source files which have been registered in the UNIT.MAP.
- To be automatically recognized as Ada source files, the files in a directory must have one of the following file extensions: .a, .ada, .adb, .ads, .bdy, .dat, .spc, .sub. You can also use any new extensions of user choice as shown in Figure 124.
- There is no naming restriction for source files explicitly registered. Source files in the UNIT.MAPof linked libraries are not contained in the current program library, but they are visible for read-only reference by the compiler and program builder.

## Generated Files

The Ada compiler output also includes files generated by the compiler or program builder. These include:

- Object module files (*.o or *.obj) and information files (*.info) for Ada source files in the program library.
- Executable files (*.exe) for main units in the program library.
- Optional listing files (*.lst).
- Optional cross reference files for use by the cross reference compiler option.
- Other intermediate files, if kept (see the -ke option).

## Ada-PAL Compiler/Library Interaction

The Ada-PAL Compiler/Ada C/C++ Changer uses the program library to locate the source files needed to handle semantic dependencies during compilation of a specified source file. Some of the situations in which this may occur are the following:

- To locate a with'ed unit or the parent unit for a separate clause or child unit
- To locate the library unit specification, if any, when a library unit body is being compiled
- To locate the body of a generic, if any, when the generic is instantiated
- To locate the body of a subprogram in another library unit to which pragma Inline applies
- To locate the body of a stub contained in a subprogram

This requires a method for locating a source file from a unit name during compilation. This is only required if the needed unit is in a source file that has not yet been analyzed in the current invocation of the compiler.

### Locating the Source File

The order of the search for locating a unit during compilation is as follows:

- First, check the UNIT.MAP of the program library;
- Then, check the UNIT.MAP of each linked library in the order of the library search list.

### Ada Program Builder/Library Interaction

The Ada program builder uses the Ada program library to locate the object module file and theinformation file for each unit needed in the main procedure. Since the names of these files are basedon the source file name, the unit-to-source correlation is required. The method used to determinethis is similar to that used by the compiler.

### Locating the Information File and Object Module

To determine the name of the source file, the program builder checks to see if a unit is registeredin the program library or any linked library. Thus, the effective order of the search by the programbuilder is:

- First, check the UNIT.MAP of the program library
- Then, check the UNIT.MAP of each linked library in the order of the library search list

Once the source file name is found, it may be that the corresponding information and object modulefiles do not exist because the source was never compiled, or they are out-of-date because the sourcefile, or some source file on which it depends, has changed. If the missing or out-of-date objectmodule belongs to a source file in one of the linked libraries, the program build will fail becauselinked libraries are read only. Otherwise, the program builder implicitly invokes the Ada-PAL Compiler/Ada C/C++ Changer to create the needed object module and information file.

### Predefined Run Time System

There is 'C' run-time sources that provides I/O, tasking, exception handling, and memory management modules which are normally required by Ada 95 language for the 'C' converted code base. These are called Ada run time system (RTS)

## Program Library Options Tool (adaopts)

### Overview

The program library options tool (adaopts) supports tailoring the program library to meet the needsof a particular Ada project. For small Ada projects, it is unlikely that this tool will be needed becausethe behavior of the compiler and builder are, by default, configured for small projects.

For more complex programs, the user may direct the program library options tool to distribute sourceto multiple directories, create links to existing program libraries, place object modules in a separatesubdirectory, etc.

The program library options tool supports the following functions:

- Creating a new program library.
- Listing all or specific values for the program library options.

- Modifying the source directory list, the library search list, the object file subdirectory, the information file subdirectory, or the cross reference file subdirectory.

Listing the source file names or library unit names registered in the program library.

### Program Library Options Tool Outputs

The program library options tool modifies the ADA.LIB file in the project directory.

### Messages

Messages and displays generated by adaopts are written to standard error.

## Source Registration Tool (adareg)

### Overview

The source registration tool (adareg) maintains the UNIT.MAP file. The UNIT.MAP file, which islocated in the project directory, contains the unit-to-source correlation of all source files thathave been registered in the program library. The source registration tool is used to register additionalsource files in the program library.

The source registration tool provides the following function:

- Explicit registration of a specified source file(s)

Registration is performed by doing a syntax analysis of a source file to determine the name and kindof the units in the file, and then adding that information to the UNIT.MAP. When the source registrationtool is invoked with a list of source files or directories containing source files, registration isperformed on all files specified on the command line. When the source registration tool is invokedwith a directory name, it registers all files with the following extensions: .a, .ada, .adb, .ads, .bdy, .dat,.spc, .sub.This is stored in your newly created Ada PAL Compiler/Ada C/C++ Changer project

### Conflicts during Registration

The following restrictions apply to source file registration:

- Two source files with the same simple file name, exclusive of the directory path, cannot be simultaneously registered.
- Two source files containing the same library unit cannot be simultaneously registered.

If either of these situations arises during source file registration, the source files are said to conflict and one of the source files and its units overrides the other, depending on whether the registration is explicit or automatic. Explicit registration of a source file, either by compiling or by the source registration tool, overrides any previously registered source files with which it conflicts. When a registered source file is overridden, it remains in the UNIT.MAP, but its units are marked as Invalid.

### Source Registration Tool Outputs

The source registration tool modifies the UNIT.MAP file in the current directory.All source registration tool messages are written to standard

# Adacgen

The Ada-Compiler translates Ada 95 source programs into relocatable object modules and records dependency information for use by the program builder. It optionally generates source listing, assembly listing and debugger information for use by the symbolic debugger. The Ada-PAL compiler consists of two phases—the front end and the back end. The front end performs syntactic and semantic analysis. It generates C source files as input to the back end. The back endof the Ada-PAL compiler is an ISO/ANSIC compiler. It performs code generation, applies optimizations, and generates a relocatable object module.

## Compiler Inputs

### Invocation

```
adacgen [option…] [file…]
```

The adacgen command invokes the Ada-PAL compiler for one or more files. If the specified source files have semantic dependencies on other units, the source files for those units must be located either in the program library or in one of the linked libraries. If a source file depends on a library unit not yet processed by the current invocation of the compiler, the compiler will find and process that library unit (through the front end only) provided that the source file containing the required library unit has been registered in the program library or is in a linked library. This proceeds recursively, if necessary, until the closure of all depended-upon library units have been processed.

## Listing Options

For the listing options, the compiler generates the requested listing for each file specified on the command line.

| Listing Options | Description |
|---|---|
| -lc* | The -lc option causes the compiler to generate a continuous source listing without pagination or headers. Any errors or other compiler-generated messages are interspersed in the listing. The listing is written to file.lst. |
| -le* | The -le option causes the compiler to generate a source listing only if there are errors. If neither -lc, -lp, or -lr are specified, the listing is generated without pagination or headers, with interspersed error messages, as if -lc had been specified. The listing is written to file.lst. |
| -lf filename | When used in conjunction with the -lc, -le, -lp, or -lr option, the -lf option causes the compiler to write the listing to filename instead of the default file.lst. |
| -lp* | The -lp option causes the compiler to generate a line-numbered listing with pagination and a header at the top of each page. The page is 60 lines long and 80 columns wide. Any errors or other compiler-generated messages are interspersed in the listing, which includes all messages generated by the compiler. The listing is written to file.lst. |
| -lr* | The -lr option causes the compiler to generate a listing containing only those lines for which compiler messages were generated, as well as the compiler |

| Listing Options | Description |
|---|---|
| | messages. The listing is written to file.lst. |
| -lx* | The -lx option causes the compiler to generate a cross reference listing. This cross reference listing is a line-numbered listing followed by a cross reference table. This listing is written to file.xlst. A binary cross reference file file.ref will also be generated. |
| -pl length* | This sets the page length for the paginated source listing to length lines. This option has no effect unless used in conjunction with the -lp option. |
| -pw width* | This sets the page width for the paginated source listing to width columns. This option has no effect unless used in conjunction with the -lp option. |
| -nh | No headers in listings. |
| * The marked Adabgen options are already added in Ada-PAL Compiler and Ada-C/C++ Changer. Do not add these in your additional options tab. If you add these options, it will break your application. | |
| **Message Options** | |
| -m msg_kind | This suppresses the display of any messages of msg_kind. |
| +m msg_kind | This enables the display of any messages of msg_kind. |
| -mrmsg_kind | This suppresses the display of any messages of msg_kind for the current invocation of the compiler and for any recursive invocation of the compiler. |
| +mrmsg_kind | This enables the display of any messages of msg_kind for the current invocation of the compiler and for any recursive invocation of the compiler. The valid values for msg_kind are:<br>• a — all messages, except that "-m a" does not suppress error messages.<br>• d— implementation-dependent messages.<br>• e— error messages.<br>• i— information messages.<br>• n— not-yet-implemented messages.<br>• w— warning messages.<br>• r— redundant messages |

By default, all messages except information and redundant messages are displayed for the current invocation of the compiler. For recursive invocations, no messages are displayed by default. For convenience, "-m a", will suppress all messages except errors.

## Other Options

| Options | Description |
|---|---|
| -0 | The -0 option identifies the version number of the executable. (That's a zero, not an oh) |
| -a | If the -a option is specified, compilation will stop after semantic analysis. No output is generated. |
| -c | If the -c option is specified, compilation will stop after the front end. No output is generated. |
| -discard_names | This option has the same effect as using pragma Discard_Names. |
| -e count | The -e count option causes the compiler to report |

| Options | Description |
|---|---|
| | only the first count errors. The default is 100. |
| -eo | The -eo option enables optimizations that are performed by the front end. This is the default. |
| -ga | Generate Ada-oriented debugging information. The –ga option causes the compiler to generate the appropriate code and data for operation with the C debugger, but in a way that should cause it to display the Ada source code rather than the C source code. |
| -gc | Generate C-oriented debugging information. The -gc option causes the compiler to generate the appropriate code and data for operation with the C debugger. This option also causes the intermediate C source files to be saved for use as program source files for the debugger, providing C-source-level debugging. |
| -help or -h | The -help option shows the different options that can be used with the adacgen command. |
| -late_inlines | The late-inlines option allows pragma Inline to be specified after a specless subprogram body. This option provides compatibility with Ada 83, and allows more aggressive inlining. |
| -N check* | Suppresses numeric checks. The check can be one of:<br>• division_check<br>• overflow_check<br>These checks are described in the RM. Using -N reduces the size of the code and increases its speed. Note that there is a related adacgen option, -s, to suppress all checks for a compilation. |
| -noeo | The -noeo option disables optimizations that are performed by the front end. |
| -noxr | The -noxr option disables generation of cross reference information by the compiler for use by a browser. This is the default. |
| -Olevel | The -O option (that's an oh, not a zero) controls the optimizations that are performed by the compiler back end. The accepted values for level are none, all, debug, 1, 2, and 3. These have the following effect:<br>• None— disable all optimizer options.<br>• All— same as -O3<br>• Debug — disable optimizations that substantially interfere with debugging. No optimizations are specified for the C compiler back end.<br>• 1— pass -O1 to C compiler back end.<br>• 2— pass -O2 to C compiler back end.<br>• 3— pass -O3 to C compiler back end.<br>If the -O option is not specified, -O1is passed to the C compiler back end (e.g. gcc). |
| -of file* | The -of option causes the compiler to read options and file names from the specified file. These are processed as though the contents of the file were on the command line. |

| Options | Description |
|---|---|
| -pB “BE options” | The -pB option passes the specified BE options to the back end. All text within the quotations is passed directly to gcc. These options precede the other options that adacgen generates and passes to gcc. |
| -prl | Record layout listing for packed record types. |
| -q* | The -q option specifies quiet mode;it suppresses all nonessential messages. |
| -rl | Record layout listing for all record types. |
| -s | The -s option suppresses all automatic runtime checking, including numeric checking. This option is equivalent to using pragma Suppress on all checks. |
| -sleh | Suppress Language Exception Handlers. If this option is specified, exception handlers that handle predefined exceptions (Constraint_Error, Program_Error, Tasking_Error, and Storage_Error) are removed from the program, if the exception is always propagated. |
| -speh | Suppress Propagating Exception Handlers. Same as -sleh, but applies to user-defined exceptions as well. |
| -suppress_aggregate_temps | This option has the same effect as using pragma Suppress_Aggregate_Temps. |
| -T | The -T option causes the compiler to report timing information for the compilation of each source file specified on the command line. |
| -v | The -v option specifies verbose mode. |
| * The marked Adabgen options are already added in Ada-PAL Compiler and Ada-C/C++ Changer. Do not add these in your additional options tab. If you add these options, it will break your application. | |

## Options for Maintainers

The following options are provided for use by maintainers of the compiler.

| Options | Description |
|---|---|
| -b | The -b option causes the message file (created by the front end) to be retained; normally it is deleted, as its contents are cryptic. |
| -f* | The -f option forces the generation of intermediate files even if the compiler finds errors. |
| -ke* | The -ke option specifies that intermediate files, which are normally deleted, are to be kept. |
| -ki* | Keep the information file generated by the compiler. The information file is generated by default except when the –a or -c option is used, or if the compilation terminates without generating an object module file. |
| -ne | The -ne option specifies that the adacgen process will not be restarted on failure. If the -ne option is not specified, the adacgen process will restart upon severe internal error such as a segment violation, bus error, or assertion failure. The process will restart with the file that was being processed when the failure occurred. If the file causes a severe error again, adacgen will restart with the next file to |

| Options | Description |
|---|---|
| | prevent infinite reprocessing of that file. |
| -nl | The -nl option specifies that the adacgen process will be restarted with the next file after the file that was being processed when the failure occurred. The default behavior without -nl is to restart with the file that caused the failure. (See also -ne.) |
| -nonr | The -nonr option specifies that the compiler front end may release any heap memory to the current heap. |
| -nz | The -nz option initializes all heap memory used by the compiler front end to a nonzero value. In hex, the nonzero value is BAD1BAD1so it is easy to spot in the debugger, and causes a Bus Error on a Sparc when it is dereferenced. |
| -pL "L options" | The -pL option passes the specified L options to the lister. |
| -t | The -t option generates a trace message as each declaration and statement is passed to the emitter phase of the front end. |
| -xB exe-path | The -xB option overrides the default back end and uses exe-path instead. |
| -xddir-path | The -xd option overrides the default ADA_MAGIC environment variable and uses dir-path instead. |
| -xL exe-path | The -xL option overrides the default lister and uses exe-path instead. |
| +bw | This displays all warning messages generated by the Ada compiler back end (e.g. by GCC). |
| * The marked options are already added in Ada-PAL Compiler and Ada-C/C++ Changer. Do not add these in your additional options tab. If you add these options, it will break your application. | |

# Compiler Outputs

## Compiler Output Files

Files produced by compilations are:

| Output Files | Description |
|---|---|
| file.info | Information recorded during compilation of a source file which is used by the program builder to determine if the object module is valid. |
| -file.o or file.obj | Relocatable object module files, one for each source file in the compilation. |

These output files are placed according to the program library parameters. Also produced are various intermediate files; these are usually deleted as a matter of course unless the -ke option is specified.

Additional files that may be produced by a compilation are:

| Output Files | Description |
|---|---|
| file.lst | Source listing if any of the -lp,-lc or -lr options are specified. |
| file.xlst | Cross reference listing if the -lx option is specified. |
| file.xref | Cross reference information in a binary format. This is for use by a browser and the cross reference lister. |

### Compile-Time Messages

All compiler messages are written to OSPAL Console View. When error messages are printed, processing does not proceed beyond the front end. No intermediate files or object code files are produced. Warning and other informational messages do not prevent further processing. The back end (i.e. C compiler) may print error messages as well; however, these will be error messages related to problems internal to the compiler itself. The option "-m a" can be used to suppress all warning and informational messages generated by the compiler. If there is an internal error in the compiler, the options -v and/or -t and/or +mr a can be used to help determine what part of the compiler contains the error; this might help you work around the problem.

The compiler may implicitly perform semantic analysis of other source files in the program library or in a linked library during an invocation in order to handle semantic dependencies on other compilation units. Compile-time messages generated during implicit processing are displayed only if the +mr option is used. Otherwise, compile-time messages are written only for processing of the source file(s) specified in the adacgen command.

## adabgen

The Ada program builder provides the facilities for creating a load module for an Ada program. It finds the object modules needed to build the executable, determines the elaboration order, and invokes the target linker to generate the load module.

In addition, the program builder implicitly invokes the compiler as needed so that all object modules are up-to-date with respect to any source files on which they depend. In fact, it is not necessary for the user to invoke the compiler directly at all — the entire program building process, including compilation, can be handled by the program builder, if desired. The load module generated by the program builder is in the format created by specified linker.

### Program Builder Processing

Program builder processing is divided into two phases — prelinking and linking. The prelinking phase handles those Ada 95 requirements that are processed at build time and identifies the list of object modules that make up the program. The linking phase invokes the target linker to combine the object modules to form a load module with all references resolved.

### Prelinking

The prelinking phase performs three functions:

- It determines the complete list of units needed for the main procedure;
- It finds or generates all object modules for the units on this list, ensuring that they are up-to date
- It determines an acceptable elaboration order.

To perform these functions, the prelinker uses the information files generated by the Ada compiler. These files contain the names of depended-upon and needed units. For a description of how the information files in the program library are found.

Finding the information files results in implicit invocations of the compiler for source files or units in the current program library if:

- The source file containing a needed unit has either never been compiled
- Or has been modified since it was last compiled

- Or a source file on which a unit depends has been modified since the unit was compiled.

The program builder uses time stamps to determine if a source file has been modified.

The prelinking phase handles all of the compilation order and completeness requirements for building the Ada program. If a part of the program is missing, or if the Ada source code contains incorrect dependencies, the prelinking phase will detect and report this.

## Linking

The linking phase of the program builder is handled by the linker. See your manual for the specified target linker for more information. The linking phase uses the default C runtime library as well as the Ada run time library.

## adabgen Inputs

## Invocation

```
adabgen [option…] [main-procedure-name…]
```

The adabgen command creates an absolute load module for the main procedure. The adabgen command must be invoked in a program library directory. If the current directory is not a program library directory, a program library is automatically created there.

The main-procedure-name must be a procedure for which the Ada source for all needed units is located either in the program library, or in one of the linked libraries. Multiple main procedures may be built in a single invocation of the builder.

**NOTE**: Donot confuse the name of the source file containing the main unit (e.g. simple.ada) with the main unit name (e.g. simple).

## Options

In addition to the options listed below, adabgen accepts all compiler options. These are applied to all invocations of the compiler that are made by the program builder.

| Options | Description |
|---|---|
| -0 | The -0 option identifies the version number of the executable. (That's a zero, not an oh) |
| -f | The -f option forces linking to occur even if there are prelinker errors. |
| -ga | Generate Ada-oriented debugging information. The -ga option causes the program builder to build an executable containing Ada-oriented debugging information. The -ga option is also applied to any implicit invocations of the compiler during program building. |
| -gc | Generate C-oriented debugging information. The -gc option causes the program builder to build an executable containing C-oriented debugging information. The -gc option is also applied to any implicit invocations of the compiler during program building. |
| -h or -help | The -help option shows the options that can be used with the adabgen command. |
| -ke* | The -ke option specifies that intermediate files, which are normally deleted, are to be kept. |
| -ll option | The -ll switch passes option to the target linker. For example, to pass "-map foo.map" to the target linker, use "-ll -map -llfoo.map". Options passed via the -ll switch follow the options to the linker that is generated by the Ada program builder. |
| -nc | The -nc option prevents recompilation. Normally, the Ada compiler is invoked by the adabgen command to recompile Ada programs as needed. |
| -nl | The -nl option prevents calling the linker. The prelinker is called but the target linker is not. |
| -no | The -no option prevents recompilations to recreate .o files that are out of date. |
| -o file | The -o option specifies the name of the output file (used instead of the default filename). |
| -ol file | The -ol file option passes file to the target linker |
| -q* | The -q option specifies quiet mode. |
| -r | Use a more "friendly" elaboration order. The default is to use an order that is more likely to fail but which can lead to more portable programs. |
| -v* | The -v option causes the program builder to print informational messages as processing proceeds. The -v option is also applied to any implicit invocations of the compiler during program building. |
| * The marked options are already added in Ada-PAL Compiler and Ada-C/C++ Changer. Do not add these in your additional options tab. If you add these options, it will break your application. | |

## adabgenOutputs

## Output Files

The Ada program builder generates a corresponding load module for eg., main-procedure-name.exe.

## Messages

All program builder messages are written to console view.

## Main features of Ada C/C++ Changer and Ada-PAL Compiler

The Ada run-time is written in Ada 95, and then translated to C/C++. The run-time is layered, and is re-hostable on various operating systems. As delivered, it depends only on C's native setjmp/longjmp, but is structured to allow re-hosting on POSIX-like OS's, or other RTOS's that have support for threads and some kind of "mutex".

**Ada C/C++ Changer & Ada-PAL Compiler** converts 100% of the Adasource into C, with no human intervention. Our tool is based on a fully validated Ada compiler, which handles the full Ada 95 language. It produces efficient and readable C that exactly matches the semantics of the original Ada program.

A single Ada source file can have any kind of code within it, though some compilers are more restrictive than that and use specific naming conventions (such as Rational's .1.ada and .2.ada, or AdaCore's .ads and .adb). Ada Tool is designed to handle any organization of code within source files. Furthermore, even though a source file might contain only a package spec, it might still have code that needs to be executed when the package is "elaborated." This code will be placed in the ".c" file for the package spec. Similarly, even though a file might contain only a package body, it might have "subunits" or "inlined" subprograms that need access to its local declarations, and so those are placed in a ".h" file for the body.

Ada-C/C++ Changer and Ada-PAL Compiler are very portable because the Ada Tool'sRTS relies mostly on the standard C run-time. However, C run-time support is not truly "real time" as it uses C "setjmp/longjmp" to accomplish multi-threading, which is not very flexible.

But by adapting the Ada Tool RTS to use the OS/POSIX Interface (Mapusoft) APIs, we can use "true" multithreading, and still be very portable to multiple O/Ss and RTOSs,

# Chapter 10. Working With Ada Tools

This chapter contains the following topics:

Creating ADA-C/C++Changer Project

Creating Ada-PAL Compiler Project

# Creating ADA-C/C++Changer Project

**NOTE**: This feature requires a license. Click http://mapusoft.com/downloads/ospal-evaluation/ to request for an evaluation license.

Ada-C/C++ Changer converts Ada 83 or Ada 95 Programs to C Source Code and keeps the C Source Code in Projects.

**NOTE1**: When you are working on 64bit architecture, make sure that -m32 flag is added to both the compiler and linker options in project properties to avoid compilation errors.

OS PAL now provides support to the following Ada features:

1. **Ada Line Count** – This feature enables you to count the Ada lines of code with a simple program. It just takes a list of file names, and prints out the numberof lines of Ada source code, lines of comments, and blank lines, counting lines ofcode the same way the license checker counts them.

   The application name is: "**ada_line_count.exe**". You have to run this .exe in cmd prompt.

   **Command**: ada_line_count file1 file2 file3 …
2. **POSIX ADA Support (For Linux only) --** Ada tools now give support to POSIX. You have a separate library with POSIX Ada packages, for Linux only. To make this "linked library" available to a given user, the adaopts command:

   **Command**: adaopts -p /usr/local/OSPAL/Tools/Ada/linux/posix_ada

   will link the posix-ada library into the "search path" for the current library.
3. **Ada Support for GNAT compatibility compiler –** This feature enables you to link the "gnat compatibility" library into the search path for the current library. The following commands are used to link:

   **Linux Command**: adaopts -p /usr/local/OSPAL/Tools/Ada/linux/gnat_compat

   **Windows Command**: adaopts -p C:\OSPAL\Tools\Ada\windows\gnat_compat
4. **Ada Support for Win32 –** This feature enables support for Ada on Win32 host**.** The following command is used for the "win32ada" library**:**

   **Windows Command:** adaopts -p C:\OSPAL\Tools\Ada\windows\win32ada
5. For Ada C/C++Changer project, from Properties page if you change Ada Main procedure, it will not build the project with that procedure immediately. You need to select the project and refresh 1-2 times and clean the project and then do the build.

**Note**: On Linux HOST/Environment, you need to set View/Modify permissions to the Tools folder while creating a project.

**To set View/Modify permissions**:

- Go to the Tools folder.
- Right click on the Tools folder, and select **Properties**>**Permissions**.
- Change the required permissions.

Then provide executable permissions to files under [tools/Ada/linux/bin] folder before creating any Ada project. Otherwise it will give an OS PAL exception while trying to convert Ada to C using Ada-C/C++Changer Options.

**To change executable permissions**:

- Go to Terminal/Command window.
- Go to the respective folder location by cd OS-PAL root directory/Tools/Ada/linux/bin.
- Once you are in "bin" folder, run the command like <chmod 777 *>
- Now observe files changes color from black to green.

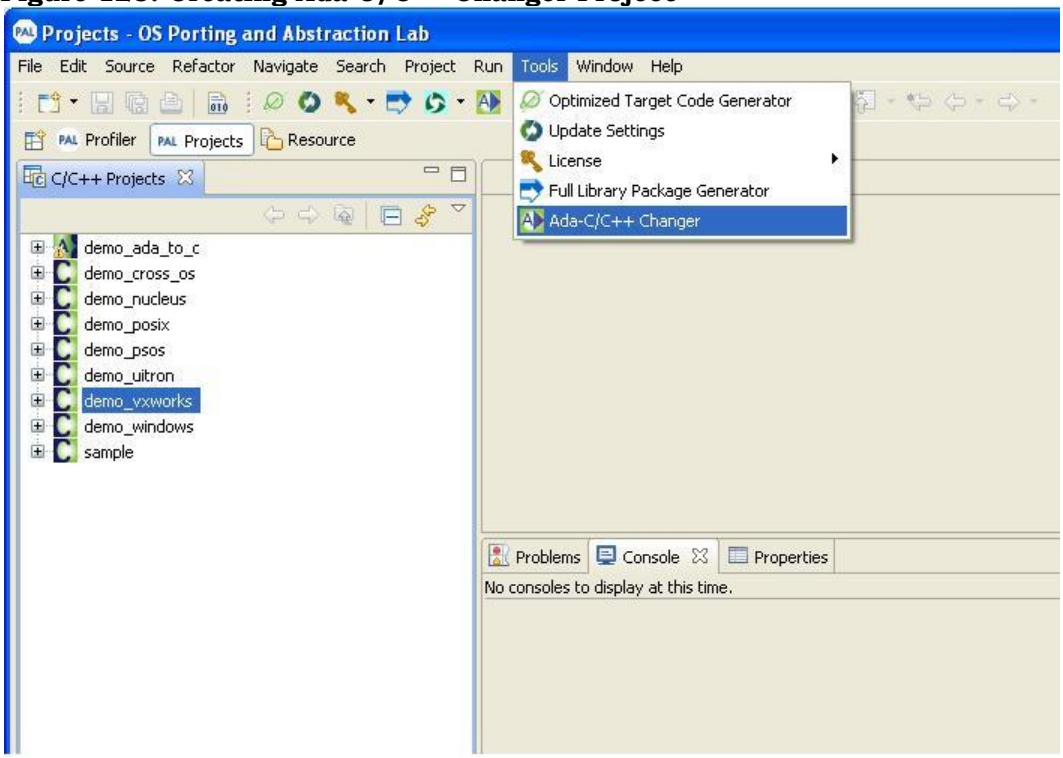
**To create Ada-C/C++ Changer project**:

1. From OS PAL main menu, select**Tools> Ada-C/C++ Changer**

   Or

   On the tool bar, click **Ada-C/C++ Changer** icon as shown in Figure 123.

**Figure 123: Creating Ada-C/C++ Changer Project**



2. On Import Ada Files page, enter the Ada project inputs as shown in Figure 124.

**Figure 124: Ada-C/C++ ChangerWindow**


**NOTE**: OS PAL now provides support to multiple Ada source directories. You can browse and select the multiple source directories on the **Ada-C/C++ Configuration Options > Additional** tab.

The field descriptions for Import Ada Files page are as follows:

| Field | Description | Your Action |
|---|---|---|
| Root Directory | Specifies the root directory which consists of Ada sources that needs to be converted to C Sources. | Select the root directory, by clicking **Browse**. |
| C/C++ Project Name | Specifies the user specific project name.<br>**NOTE**: If the project name is the same name as an already existing project, then it is considered that the user wants to **Ada Build** the project. | Enter a name to the project you want to create.<br>**Note 1:** The project name should not be more than 256 characters.<br>**Note 2:** Please avoid creating an eclipse workspace in a deeply nested sub-directory. |
| Main Ada procedure | Specifies the main procedure name of the project the user imports that is converted to the main C function that will be started as thread in Cross-OS or other interfaces. | Select the check box to enter the main Ada procedure name. For example: Sudoku_Test.<br>**Note:**To import a library project use the *-all* option for themain procedure. |
| Enter Ada File Extensions used in your Ada Project | Specifies the source files that have extensions other than the default extensions listed in the drop down list. The default extensions listed here are: .a, .ada, .adb, .ads, .bdy, .dat, .spc, .sub | You can do any one of the following:<br>• To add a new extension other than the default ones, enter in the text box and click **Add**.<br>**Note**: By default, this is enabled.<br>• To remove an extension, select the extension from the drop down list and click **Remove**.<br>**Note1:** Default extensions already available cannot be removed<br>**Note 2**: Ada Extensions are case sensitive |
| Specify Option File | Specifies if the user wants to specify any set of options that are needed for the Ada-C/C++Changer.<br>**Note**: This can be created using “space” as delimiter. | To specify an option file, select the check box and click **Browse** and select the option file.<br>**Note**: If option file is specified then Ada configuration options page will open with the specified options in the option file.<br>If not specified, then Ada Configuration options page opens with the default options. User can select or over ride the options in this page. |

**NOTE**: If you create your Eclipse Workspace in a deeply nested subdirectory, you will get an error while creating a project.

# ADA-C/C++Changer Configuration Options

On the Ada-C/C++Changer Configuration Options page, you can set the following configurations:

- Ada Source
- C/C++ Output
- Ada Listings
- Ada Messages
- Ada Drivers
- Additional

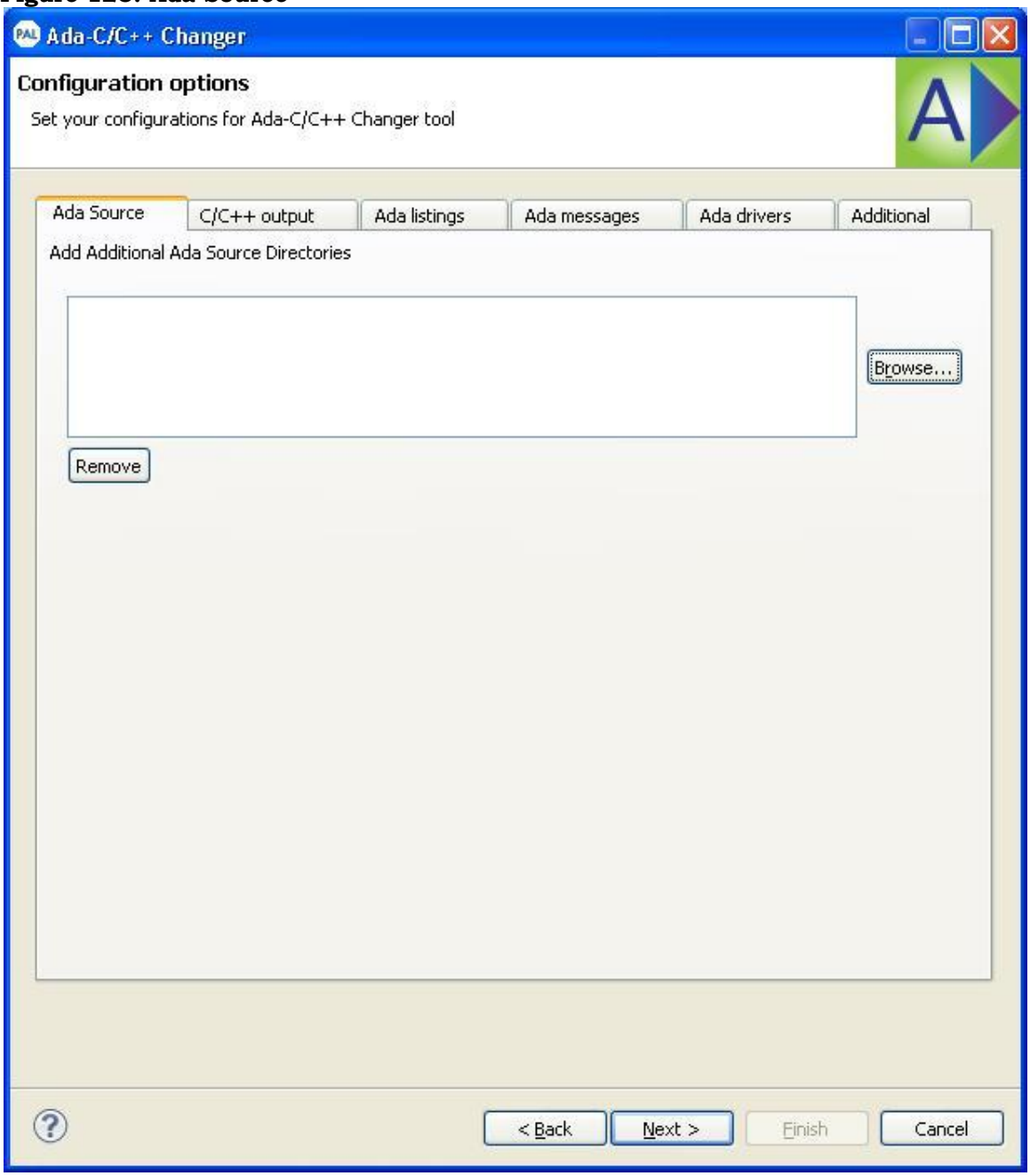
**NOTE 1**: You can change the configuration options on the Ada-C/C++Changer Property Page. To go to the Ada-C/C++Changer Property Page, right click on the project and select **Properties**.

**Ada-C/C++ Changer Options Configurations File:** Ada-C/C++ Changer options are configurable via an Option's file.

a. Once you have an Ada-C/C++ Changer project and you want to use the same options that you have used.
b. Browse to the workspace directory location and into your project_name directory.
c. You will find a file called "options" file and you can save that file in a different location (you can also rename it if you want) and use it again and again.
d. You can create new Ada-C/C++ Changer projects by passing the file info in the first screen. Ada-C/C++ Changer reads this info and sets up the GUI configuration values accordingly.
e. This way you can create an option file and use it repeatedly as a template. However, if you later want to modify these options after creating the project, you can select the Ada-C/C++ Changer project and right click and choose **Properties** and select **Ada-C/C++ Changer Configurationpage** and change. This will get stored as the new option file for that project. This gives you the flexibility to use the template when you create the project and also let you change if needed.

3. On Ada Source tab page, add the Ada source directories as shown in Figure 125.

**Figure 125: Ada Source**



The field descriptions on Ada Source tab are as follows:

| Field | Description | Your Action |
|---|---|---|
| Add Additional Ada Source Directories | Specifies if you want to add any additional Source directories. | You can do any one of the following:<br>• To select the additional Ada C/C++ Changer source directories, click **Browse** and select the Ada source directories.<br>• You can also remove an Ada C/C++ Changer source directory by clicking **Remove**. |

4. On C/C++ Output tab page, describe the C Source Options as shown in Figure 126.

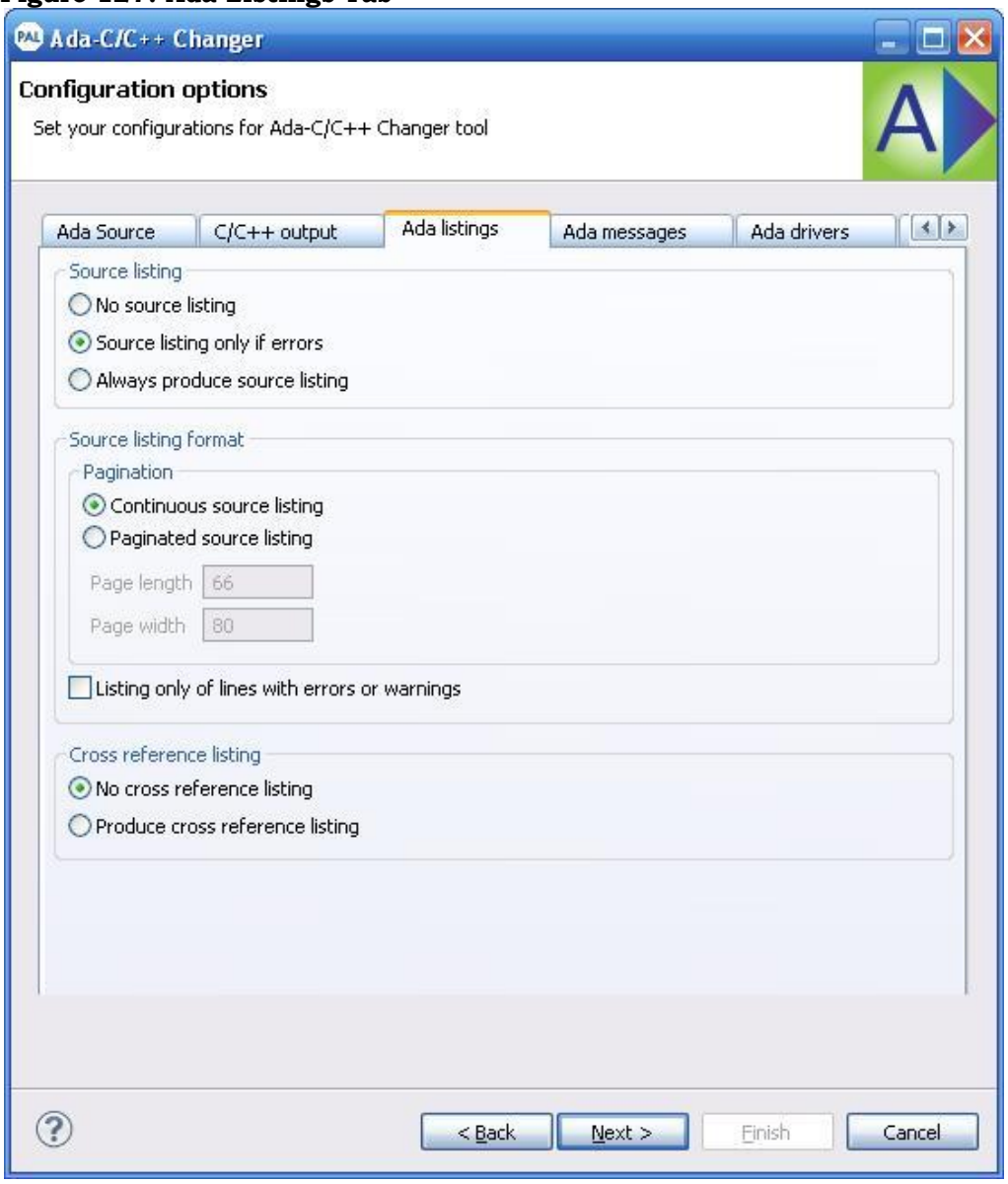
**Figure 126: C/C++ Output Page**

The field descriptions on C/C++ Output tab are as follows:

| Field | Description | Your Action |
|---|---|---|
| Comments | Specifies if you want the comments to be generated in C Source Files. | You can do any one of the following:<br>• To include comments in generated C code, select the radio button.<br>**Note**: By default, this is enabled.<br>• To suppress comments in generated C code, select the radio button. |
| Include C/C++ | Specifies if you want to include C/C++ files. | You can do any one of the following:<br>• To include full C-output only, select the radio button.<br>**Note**: By default, this is enabled.<br>• To include C++ namespace/Exceptions, select the radio button. |
| Header Files | Specifies if you want the header files to be separated or included in the C files itself. | You can do any one of the following:<br>• To separate .c and .h files in generated C Source files, select the radio button.<br>**Note**: By default, this is enabled.<br>• To not to generate separate .h files in generated C Source files, select the radio button. |
| Association | Specifies the association constructs. | You can do any one of the following:<br>• To place "{" and "}" on same line as associated construct, select the radio button.<br>**Note**: By default, this is enabled.<br>• To place "{" and "}" on its own line, select the radio button. |
| Checks | Specifies if you want to select the corresponding checks needed. | You can do any one of the following:<br>• To suppress all checks in generated C code, select the radio button.<br>**Note**: By default, this is enabled.<br>• To suppress numeric overflow checks in generated C code, select the radio button.<br>• To include all run-time checks in generated C code, select the radio button. |
| Limit on the length of the generated C Source Line | Specifies the length of the line in the generated C Source files.<br>You can change the length as required. | Enter a value to specify the length of the line in the generated C Source files.<br>**Note**: The default value is 80. |

5. On Ada Listings tab, set your listing options as shown in the Figure 127.

**Figure 127: Ada Listings Tab**

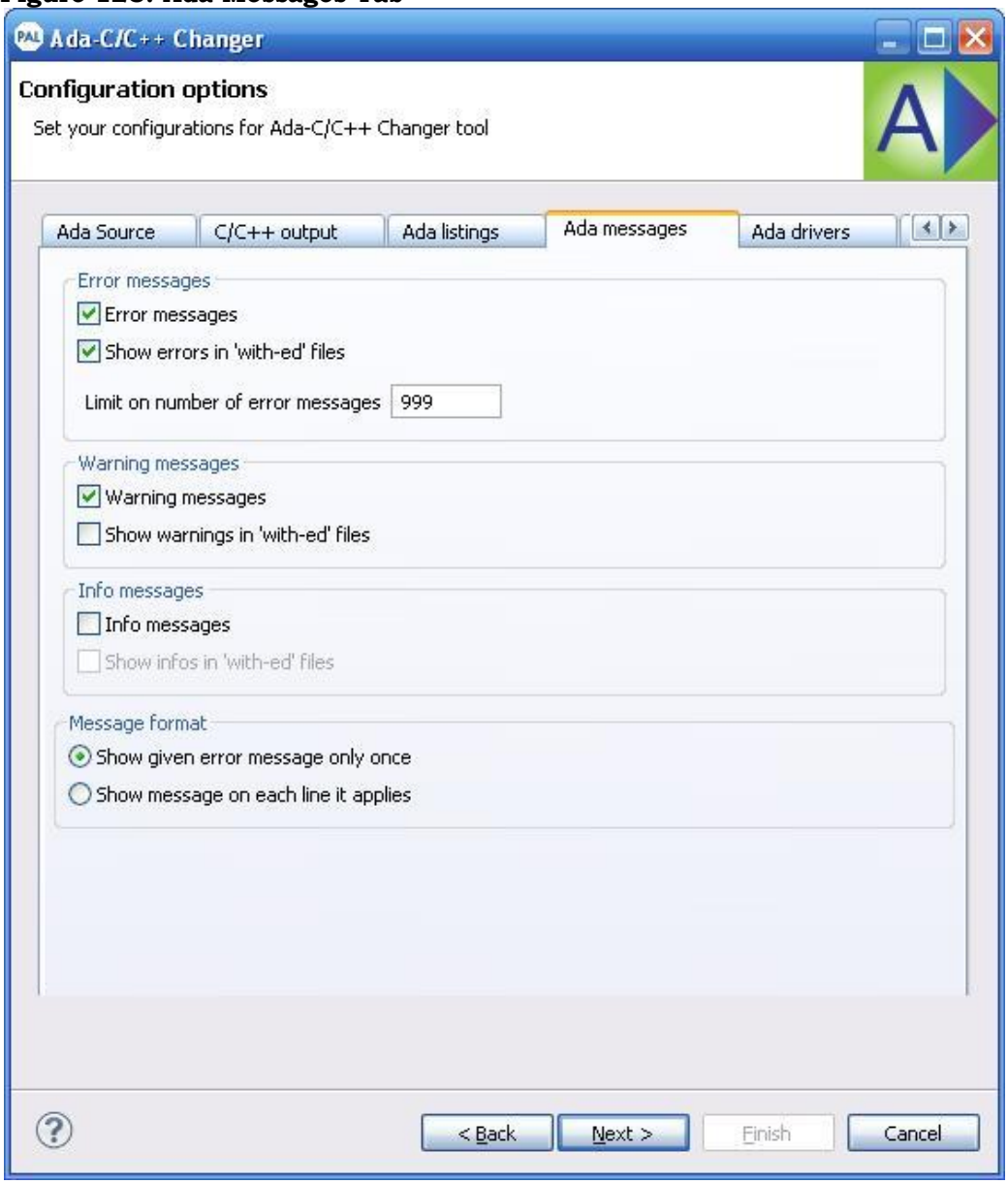


The field descriptions on Ada Listings tab are as follows:

| Field | Description | Your Action |
|---|---|---|
| Source Listing | Specifies how you want the Ada source list to be generated or not. | You can do any one of the following:<br>• To not to generate Ada source list, select the radio button.<br>• To do Ada source list only if errors are present, select the radio button.<br>**Note**: By default, this is enabled.<br>• To always produce Ada source list, select the radio button. |
| Source Listing Format | | |
| Pagination | Specifies the format of the Source Listing. | You can do any one of the following:<br>• For continuous source listing, select the radio button.<br>**Note**: By default, this is enabled.<br>• For listing only of lines with errors or warnings, select the radio button.<br>• For paginated source listing, select the radio button. |
| Page Length | Specifies the length of the page. | Enter a value for the length of the page.<br>**Note**: By default, the value is 66. |
| Page Width | Specifies the width of the page. | Enter a value for the width of the page.<br>**Note**: By default, the value is 80. |
| Listing only of lines with errors or warnings | Specifies if you want to list only lines with errors or warnings | Select the check box. |
| Cross Reference Listing | Specifies if you want to generate a cross reference listing or not. | You can do any one of the following:<br>• To not to generate a cross reference listing, select the radio button.<br>**Note**: By default, this is enabled.<br>• To generate a cross reference listing, select the radio button. |

6. On Ada Messages tab, set your message options as shown in Figure 128.

**Figure 128: Ada Messages Tab**

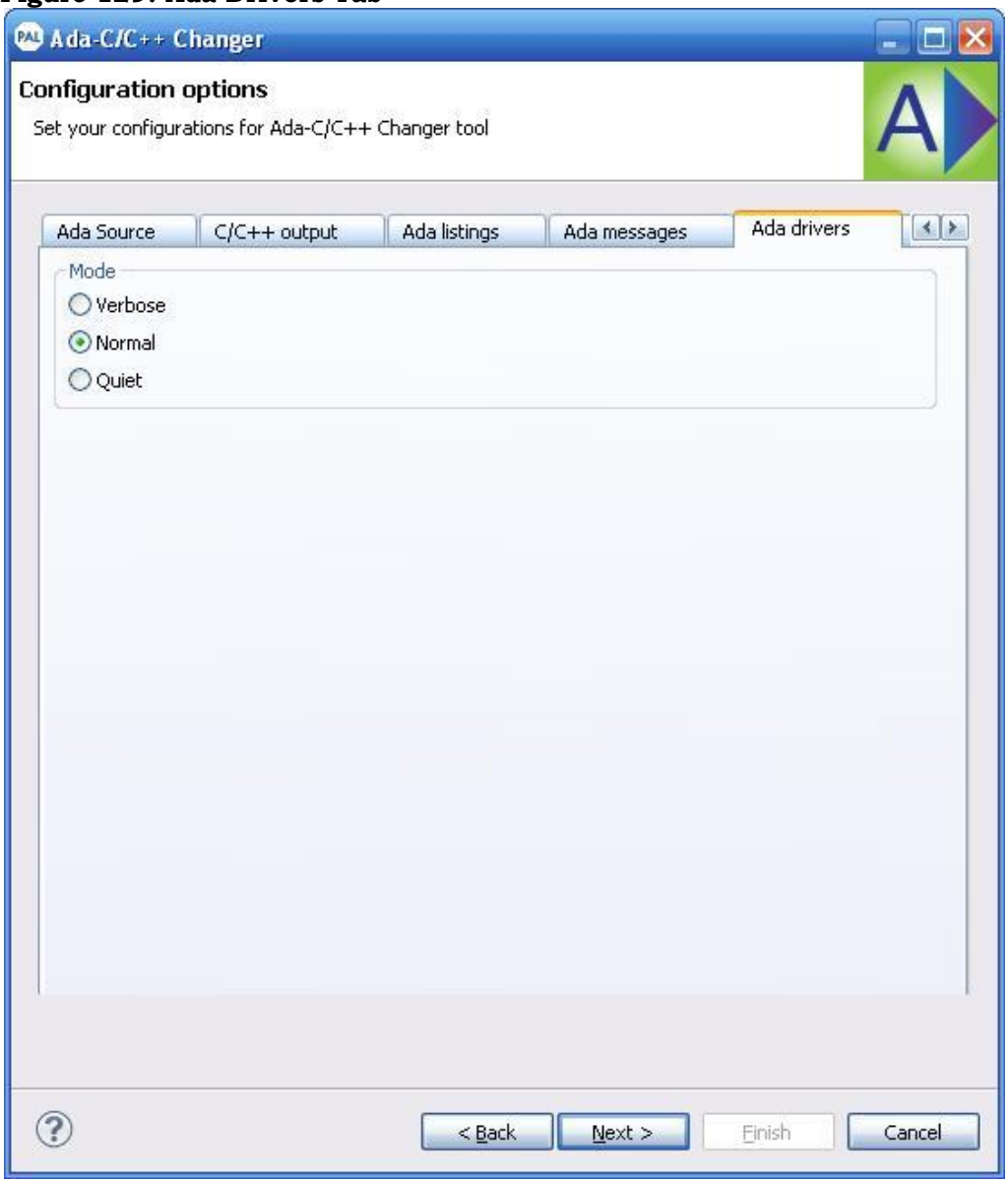


The field descriptions on Ada Messages tab are as follows:

| Field | Description | Your Action |
|---|---|---|
| Error Messages | Specifies if you want to select error messages. | You can do any one of the following:<br>• To select error messages, select the check box.<br>**Note**: By default, this is enabled.<br>• To show error sin with "ed" files, select the check box. |
| Limit on number of error messages | Specifies the count of error messages. | Enter a value to specify the limit on number of error messages.<br>**Note**: By default, the value is 999. |
| Warning Messages | Specifies if you want to select warning messages. | You can do any one of the following:<br>• To select warning messages, select the check box.<br>**Note**: By default, this is enabled.<br>• To show warnings in with "ed" files, select the check box. |
| Info Messages | Specifies if you want to select the info messages. | You can do any one of the following:<br>• To select info messages, select the check box.<br>• To show infos in with "ed" files, select the check box. |
| Message Format | Specifies the format of the message. | You can do any one of the following:<br>• To show given error message only once, select the radio button.<br>**Note**: By default, this is enabled.<br>• To show message on each line it applies, select the radio button. |

7. On Ada Drivers tab, set your driver options as shown in Figure 129.

**Figure 129: Ada Drivers Tab**

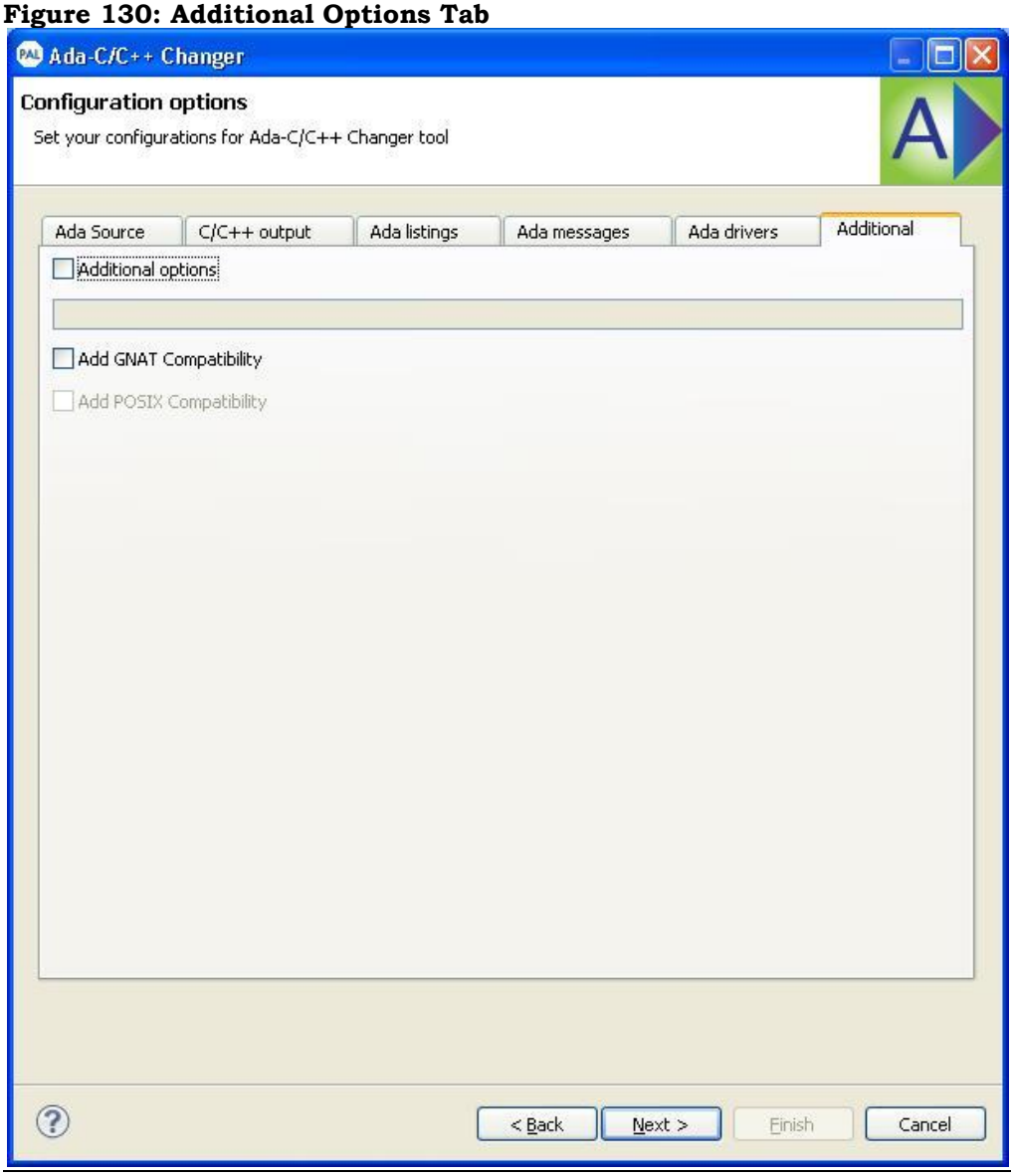


The field descriptions for Ada driver options tab are as follows:

| Field | Description | Your Action |
|---|---|---|
| Mode | Specifies the required mode for reporting the compiler actions. | You can do any one of the following:<br>• To select the verbose mode, select the radio button.<br>• To select the normal mode, select the radio button.<br>**Note**: By default, this is enabled.<br>• To select the quiet mode, select the radio button. |

8. On Additional Options tab, set your miscellaneous options. You can also select the multiple Ada source directories and click **Next** as shown in Figure 130.

**Figure 130: Additional Options Tab**

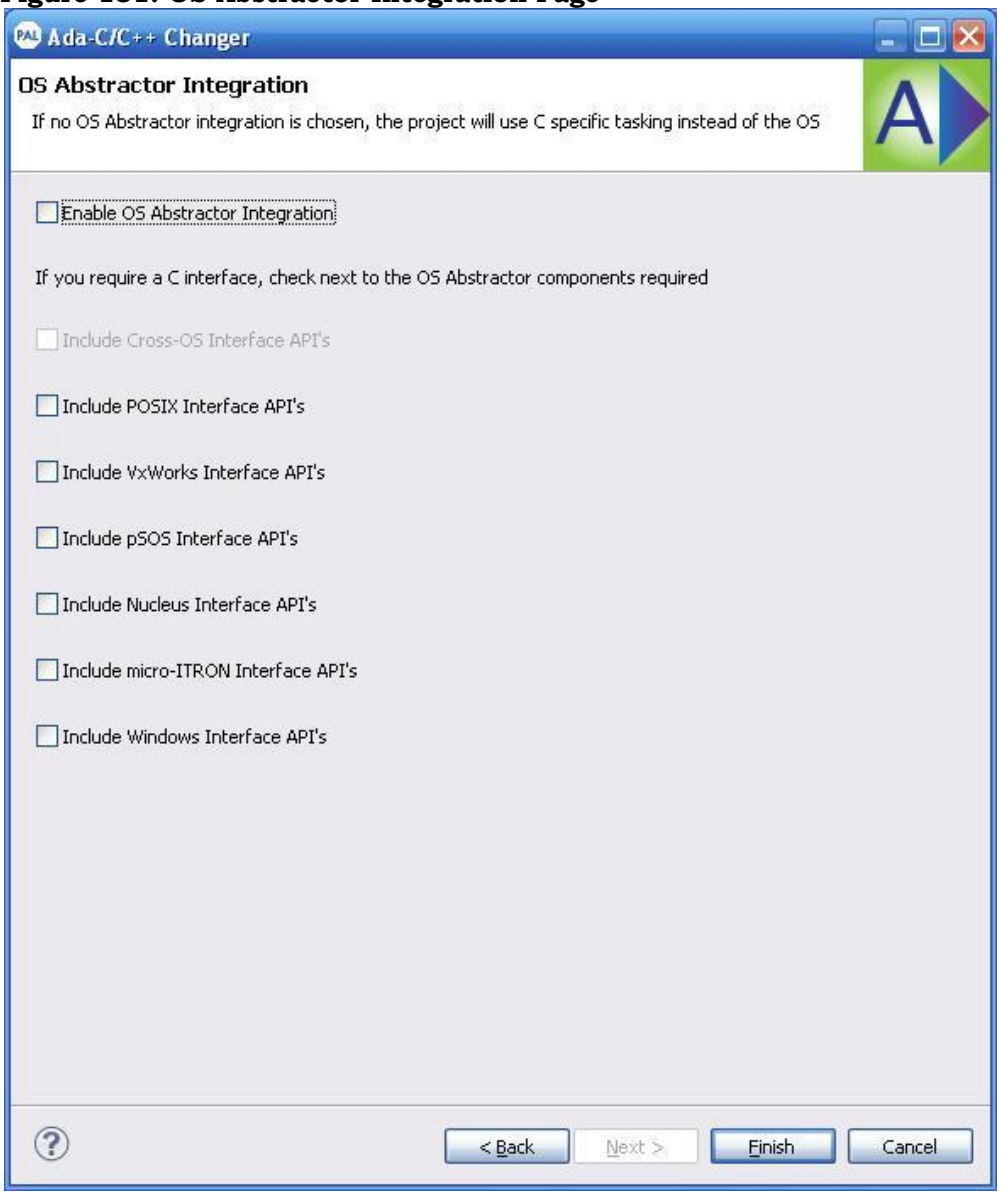


The field descriptions for additional options tab are as follows:

| Field | Description | Your Action |
|---|---|---|
| Additional Options | Specifies if you want to include any other additional options such as custom or optional. | To specify additional options, select the check box and enter a value in the text box. |
| Add GNAT Compatibility | Specifies if you want to add GNAT Compatibility. | To add GNAT compatibility, select the check box. |
| Add POSIX Compatibility | Specifies if you want to add POSIX Compatibility.<br>**Note**: This feature is not supported on Windows. It is supported on Linux only. | This feature is disabled on Windows.<br>On Linux, to add POSIX compatibility, select the check box. |

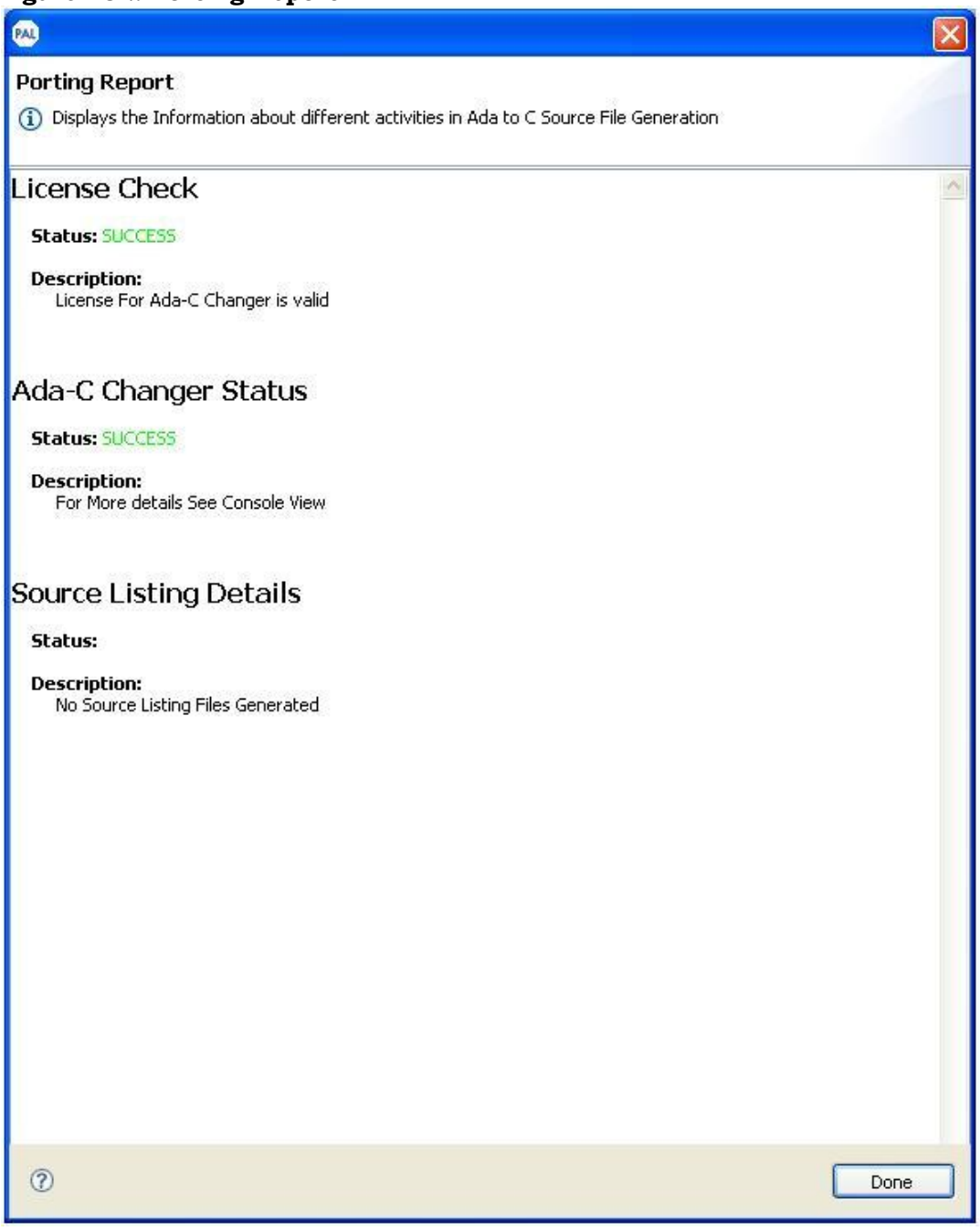
9. On OS Abstractor Integration page, select the check box to enable OS Abstractor integration. To link the OS Abstractorcomponents to the generated C Source Code, select the check box for the corresponding interfacesupport from the available list and click **Finish** as shown in Figure 131.
   **NOTE 1**: If no OS Abstractor Integration is chosen, the project will use C specific tasking instead of the OS.This will not allow you to migrate to multiple Operating Systems and you cannot enable OS Abstractor Integrationfor this project after importing to OS PAL.
   **NOTE 2:** If you have enabled the Cross-OS APIs, you can always enable the additional development APIs after importing to OS PAL.

**Figure 131: OS Abstractor Integration Page**



10. After porting Ada code into C Source Code is successfully done, the porting report page is displayed as shown in Figure 132. Click **Done** to complete the process. The report gives detailed information on the status of different activities in Ada to C source file generation.

**Figure 132: Porting Report**




You have now successfully created Ada-C/C++Changer project.

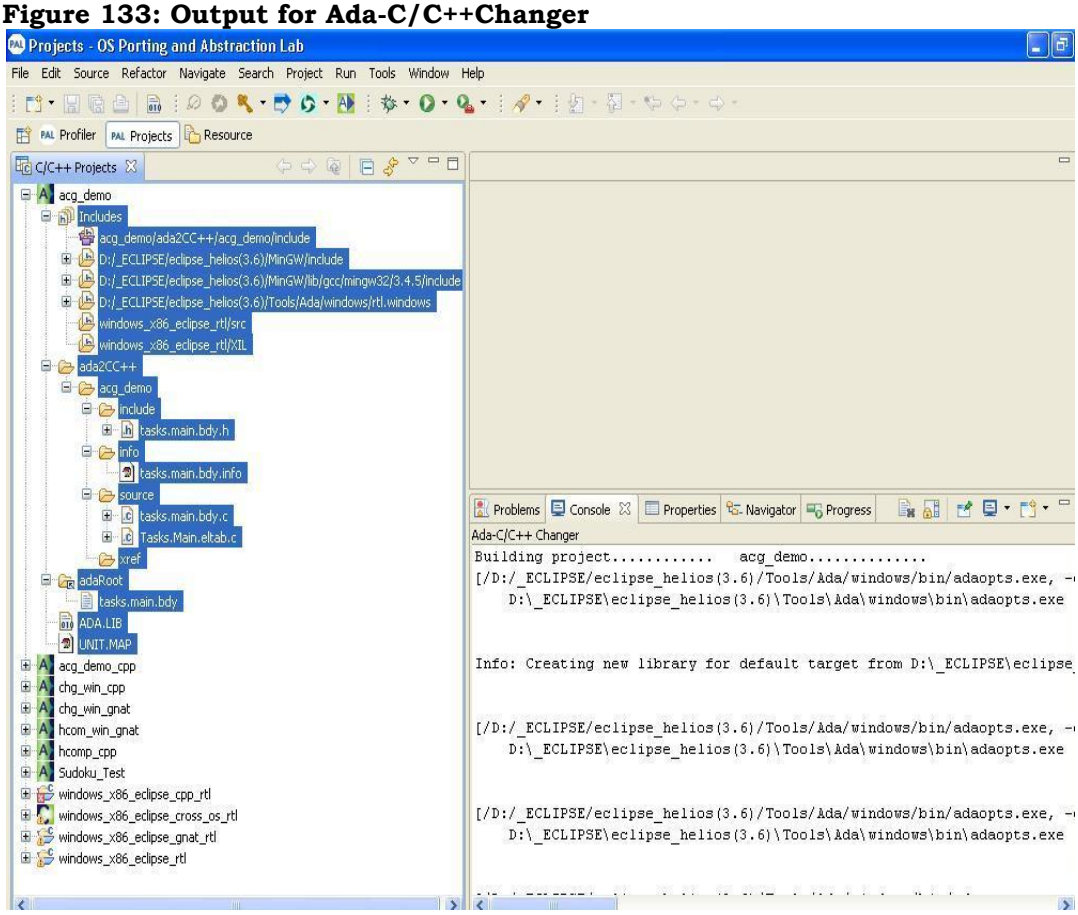
**NOTE:** While importing an Ada project, you will receive 3617 warning messages. It appears that most, if not all of them, are associated with the rtl files.

They are as follows:

- Defined but not used variables(3129)
- Return with no value, in function returning non-void (15)
- Assignment from incompatible pointer type (52)
- Comparison is always false due to limited range of data type (94)
- Cast from pointer to integer of different size (3)
- Comparison of distinct pointer types lacks a cast (8)
- Control reaches end of non-void function (71)
- Implicit declaration of function (10)
- Initialization from incompatible pointer type (5)
- Integer constant is so large that it is unsigned (1)
- Left shift count >= width type (1)
- Missing braces around initializer (2)
- Passing arg from incompatible pointer type (114)
- Statement with no effect (28)
- Unused variable (83)
- This decimal constant is unsigned only in ISO C90 (1)

11. To view the C/C++ code, expand the project you have created as shown in the Figure 133.

**Figure 133: Output for Ada-C/C++Changer**



The Ada-C/C++ Changer project has the following files:

- **Includes**–This folder contains the header file paths of your project
- **Ada2CC++**–This folder contains all the files generated by Ada tools
  - **Ada C/C++ Project name**–This folder contains the Ada-C/C++ project related files
    - **Include**–This folder contains the Ada Include folder
    - **info–**This folder contains information file subdirectory of the program library directory where information files for the object modules are placed.
    - **source**–This folder contains the converted C/C++ source files. If OS Abstractor is integrated, then you get a folder, **init**, which contains the OSPAL template files.
      **NOTE:** For more information on the Template files, refer to OS PAL Template Files section in this manual
    - **xref–**This folder contains the cross-referenced files which are generated by the Ada tools
- **adaRoot–**This folder contains the Ada sources added during your project creation. In case, you add any additional sources, you can view this in the **Project > Property page >Ada Source** tab of the respective project.
- **ADA.LIB–**This contains information describing the configuration of the Ada library
- **UNIT.MAP**–This contains a unit-to-source mapping for use by the compiler and program builder
- **.options**–This contains the list of options with which Ada-C/C++ project or executable is created. This is a hidden file. You can view this in Navigator view. To view select **Window > Show View > Other > General > Navigator**.

**NOTE**: Host Libraries and include paths are automatically added during project creation. For viewing this information, select **Ada-C/C++ Changer Project > Properties > C/C++ Build > Settings**.

## Ada-C/C++Changer Build

Ada-C/C++ Changer enables you to build an existing project. This feature enables you to either do an incremental full build or just a C/C++ Changer Build on your Ada 95 sources.

- The Build process will incrementally compile the Ada files that have been modified or added since the last build.

  **NOTE**: To do an incremental build, you should not do **Clean**.

- If any new Ada source files are added, removed, or modified in the project, and want to generate the c-sources again, you can do a full build by first calling **Clean** and then **Build.**

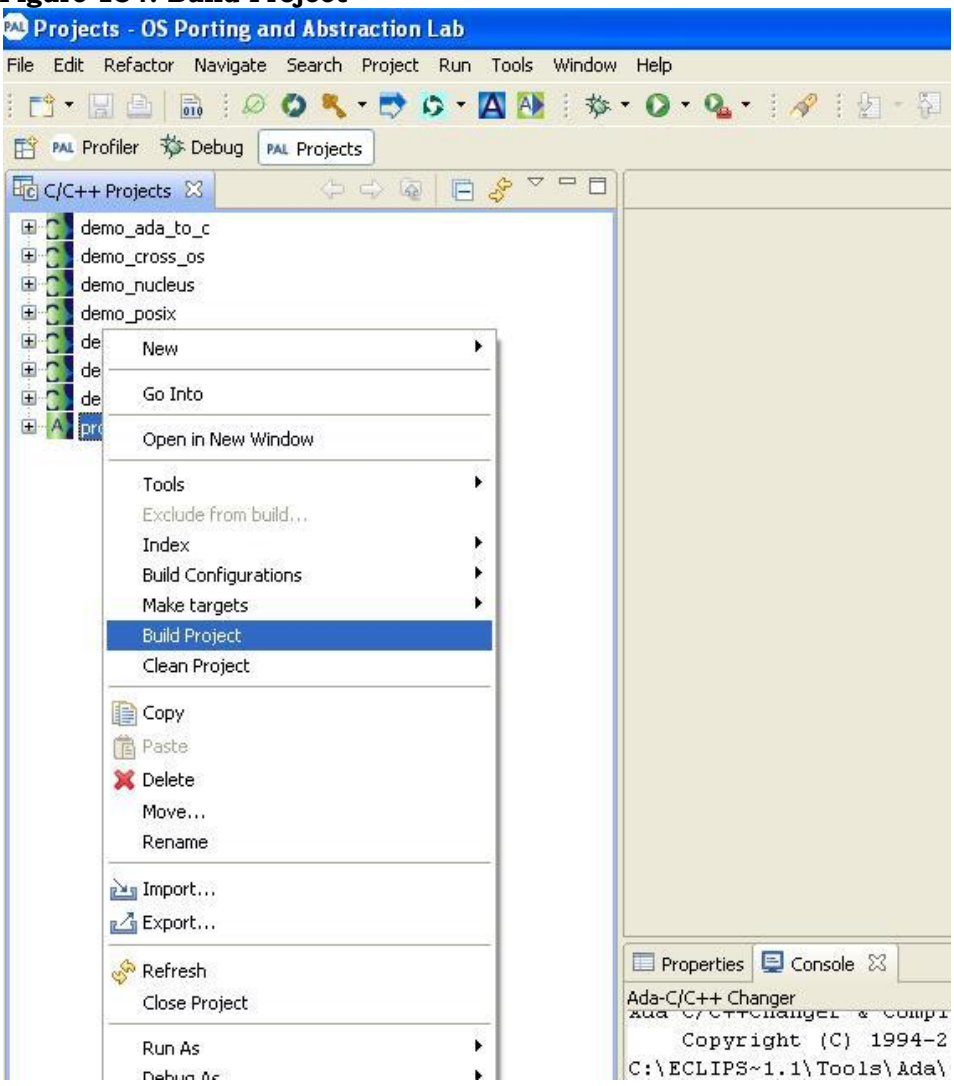
**NOTE: Clean** will delete all your info files, which will result in a full build.

- You cannot re-build if new directories are created or new files are added with differing extension than what was provided during the project creation. If you have new directories and new extensions, then you must recreate the Ada-C/C++ Changer project.
- You will get a build error when you create an Ada-C/C++ Changer project with the default "–all" option for the Main Ada Procedure Name.

**To do Ada-C/C++ Changer Build:**

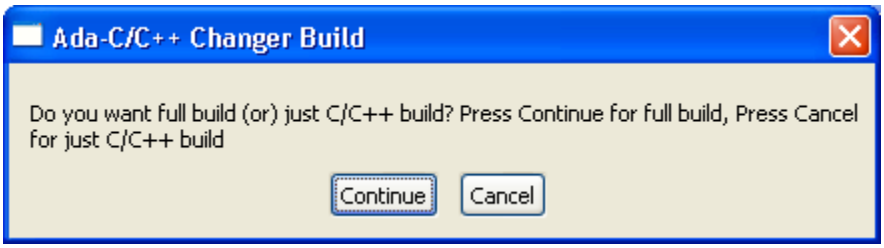
1. To generate the corresponding executable, right click on the project you have created on the projects pane, and select **Build Project** in project or from the main menu select **Projects**>**Build project** as shown in Figure 134.

**Figure 134: Build Project**



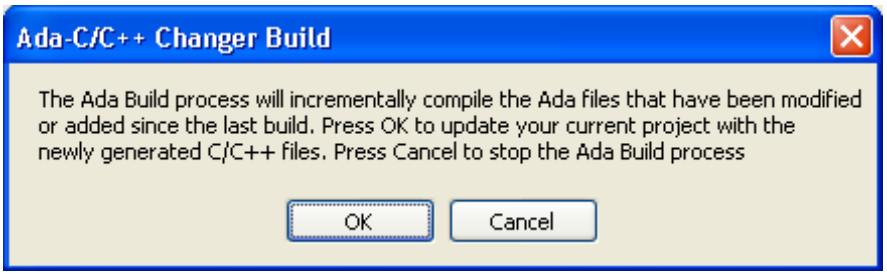
2. A pop-up window is displayed asking if you want to do a Full Build or just a C/C++Build as shown in Figure 135. Full Build includes Ada Build, which will incrementally compile the Ada files that have been modified or added since the last build, and C/C++ build. Click **Cancel** for doing a C/C++ build.

**Figure 135: Ada-C/C++ Changer Build**



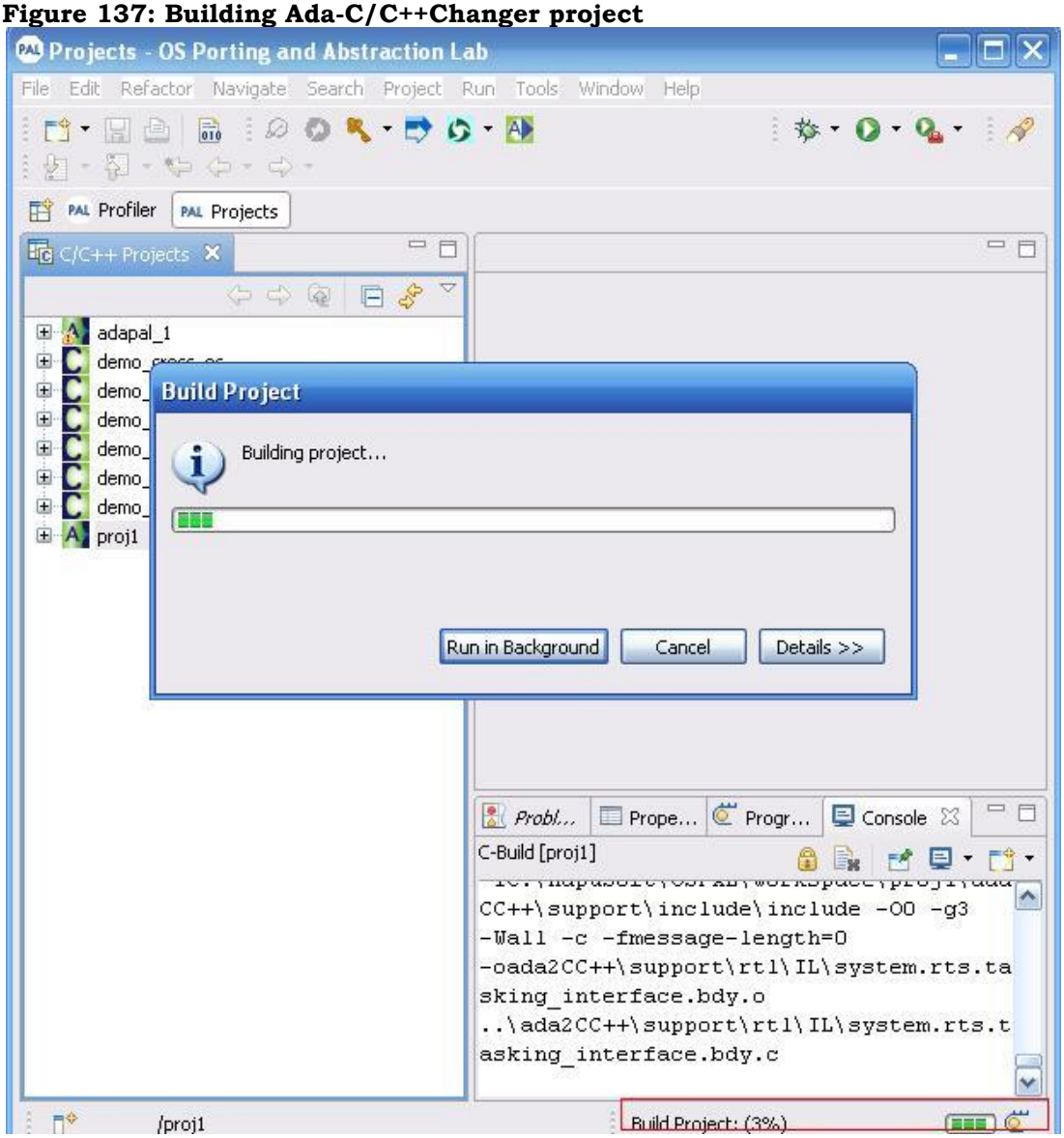

3. If you click **Continue**, another pop-up window is displayed. Click **OK** to complete the Full build or click **Cancel**to stop the Ada Build process as shown in Figure 136.

**Figure 136: Ada Full Build**



4. The Ada-C/C++Changer project starts to build and generates the .exe file as shown highlighted in Figure 137.

**Figure 137: Building Ada-C/C++Changer project**

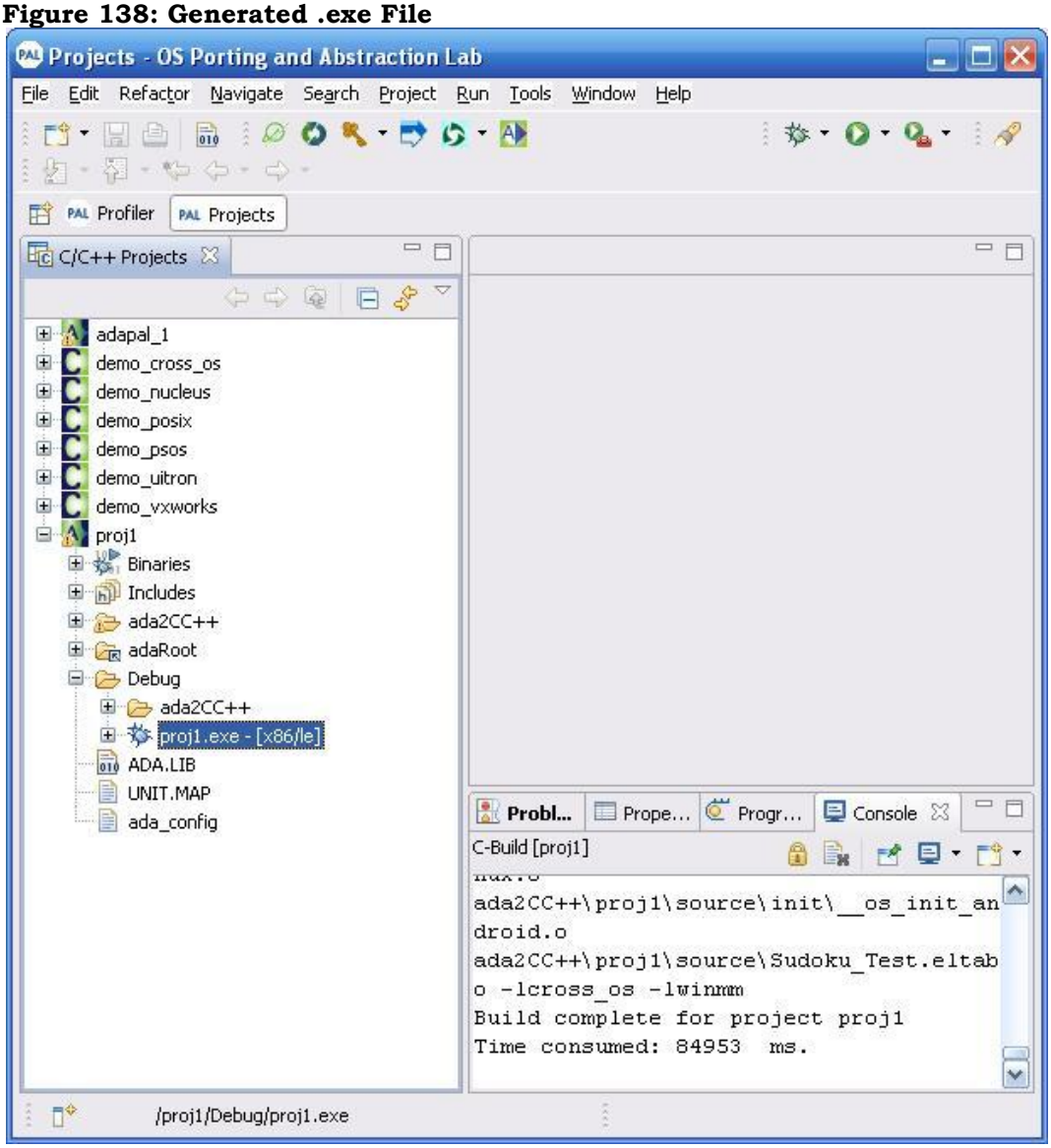



**NOTE**: While running any Ada project after build, the project sometimes will again build the project before running the application. To avoid this do the following configuration:

- Select **Window > Preferences > Run/Debug > Launching.**
- Under **General Options,** deselect the check box for **Build (if required) before launching** flag.

5. You can view the generated .exe file under the project in the **Debug**Folder as shown in the Figure 138**.**

**Figure 138: Generated .exe File**




**NOTE 1:** When you build Ada-C/C++ project, you may get many warnings.

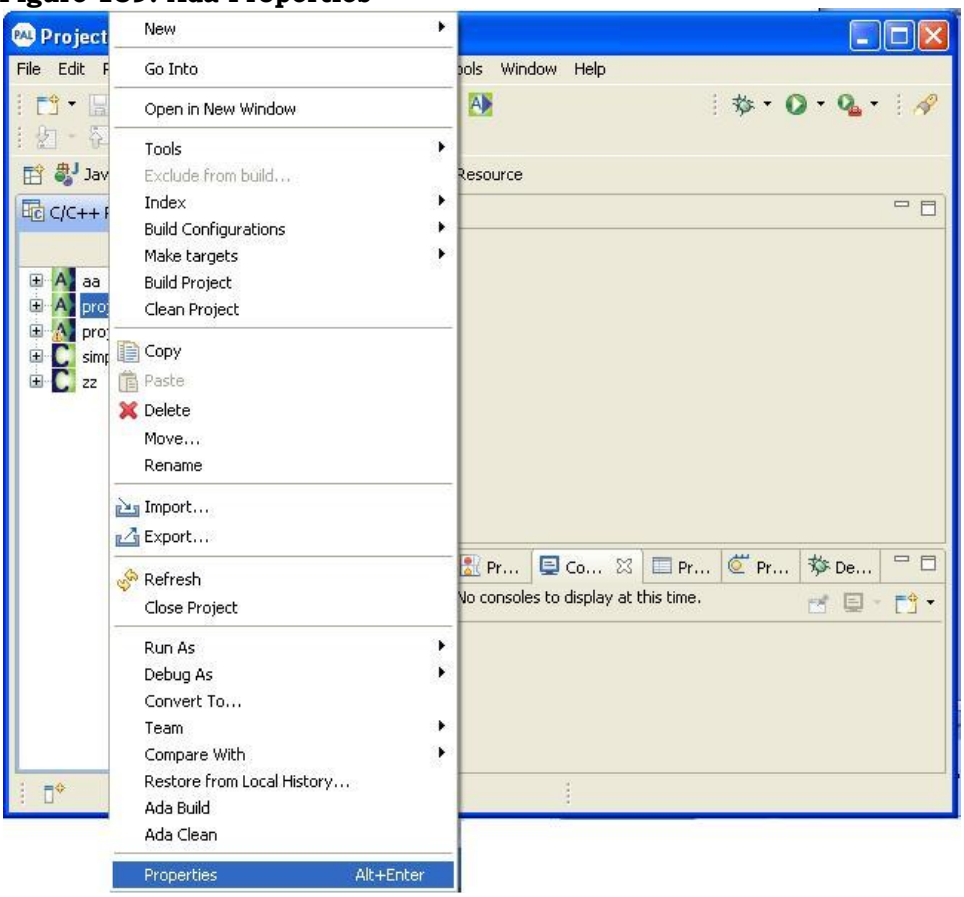
## Ada-C/C++Changer Property Page

Ada-C/C++Changer Property Page enables you to change or modify the configuration options you have set for your project.

To go to the property page:

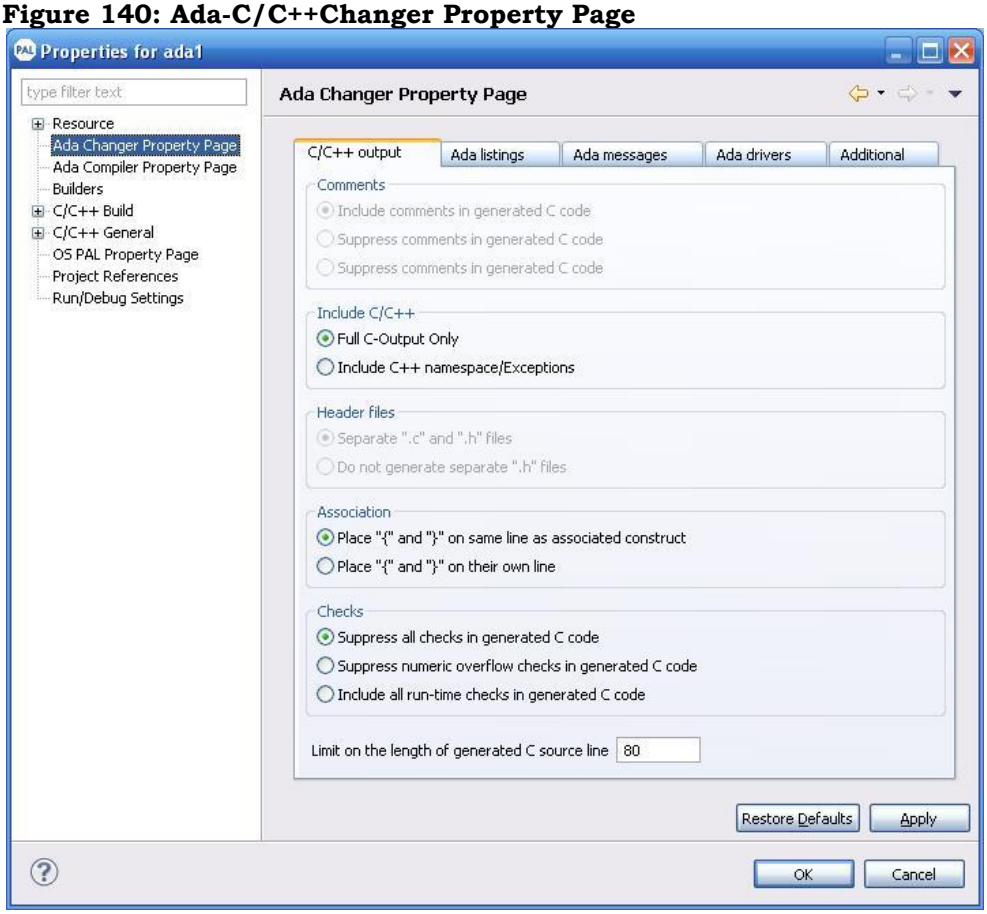
1. On OS PAL Projects pane, select the Ada-C/C++Changer project you have created. Right click on it and select **Properties** and shown in the Figure 139.

**Figure 139: Ada Properties**



2. The Ada-C/C++Changer property page is displayed as shown in the Figure 140. Make the necessary changes and click **Apply**.

**Figure 140: Ada-C/C++Changer Property Page**




When you do an Ada Build, the project will build the sources according to the recent changes you have made to the Ada Configuration Options.For more information about Ada Configuration Options, refer to Ada-C/C++ Changer Configurations Options section.

**NOTE**: For Ada-C/C++Changer project, from Properties page if you change Ada Main procedure, it will not build the project with that procedure immediately. You need to select the project and refresh 1-2 times and clean the project and then do the build.

## Target Code Generation for Ada-C/C++ Changer Projects

OS PAL allows only Target Code Generation for Ada-C/C++ Changer Projects, when the projects are created with OS Abstractor Integration. Ada-C/C++ Changer projects cannot be code-optimized.

For Ada-C/C++ Changer projects created without OS Abstractor Integration, Target Code Generation should not be done. The generated code WILL NOT work since there is no call to OS_Application_Initialize.

For Ada-PAL Compiler Projects, Cross-OS interfaces are added directly to the project as target sources, if you have a valid and relevant Standalone Package Generator license. If Target Code Generation is attempted on these projects, all the Cross-OS functionality, being part of application, is again redefined in cross_os.c. This will give re-definition errors on compile time.

Hence Ada-PAL Compiler projects cannot be code-optimized.

**NOTE**: For Ada-C/C++ changer projects along with Abstractor, if you do target code generation, it will generate sample project files. You have to generate your own project files to generate binaries.

## Manual Modifications to Projects files generated by Target Code Generator

The target code output produced when optimizing Ada projects via the target code optimization process is a little different than that of the standard C/C++ OS PAL project. In this case, the API level optimization process is skipped as the application needs to link-in other required RTL C/C++ libraries and possible other 'C' interface libraries. Instead of the OS Abstractor code being included as part of the application, it is added into the target directory as a separate code base that should be built as libraries. There will be separate libraries for the Cross OS component as well as any other OS Interface components (like VxWorks, Windows, etc.) included in the project. There will also be a separate Ada Run Time Library(RTL) required to be linked in as well.

For example, if a converted Ada to C/C++ project that was integrated with OS Abstractor and includes the POSIX Interface were optimized for a windows target it would look like follows:

```
<target dir>
    cross_os_windows
        source
        include
        specific
    posix_interface
        source
        include
        specific
    include
        include
    rtl
        XIL
        src
        IL
```

```
<app name>
    ada2C++
        <app name>
            source
            include
```

The <target_dir> is the directory location where the generated code would be placed. The Cross OS and OS Interface directories will include project files specific to your target. Project files for the RTL will only be included for Windows and Linux targets. On a Windows target, you will get a project file for the Eclipse IDE and on Linux you will get both Eclipse and make files. For all other targets and toolsets you will need to create an RTL library project. An application project will be created for the target, but it will require some manual modifications to build.

The modifications which need to be made to the application project are as follows:

**Header Inclusion**

```
    Add include paths for the Ada RTL component.
        <target dir>/rtl
        <target dir>/rtl/src
        <target dir>/rtl/IL

    Add include path for any other 'C' library that you need for your
application

    if your Ada project is integrated with Cross OS you will need to
add the following:
        <target dir>/include/include
        <target dir>/cross_os_<target>/include
```

# Creating an Ada-PAL Compiler Project

## Compiling Ada C Code

Ada-PAL Compiler converts Ada Program Library directly into executable binary file.

For Ada-PAL Compiler Projects, Cross-OS interfaces are added directly to the project as a target sources if user has the relevant Standalone package generator license. If a Code Generation is attempted on these projects, all the Cross-OS functionality being part of application is again redefined in cross_os.c. This will end up in having re-definition errors on compile time.

Hence Ada-PAL Compiler projects cannot be code-optimized.

**NOTE:** When you are working on 64bit architecture, make sure that -m32 flag is added to both the compiler and linker options in project properties to avoid compilation errors.

OS PAL now provides support for the following Ada features:

1. **ADA Line Count** – This feature enables you to count the Ada lines of code with a simple program. It just takes a list of file names, and prints out the numberof lines of Ada source code, lines of comments, and blank lines, counting lines ofcode the same way the license checker counts them.

   The application name is: "**ada_line_count.exe**". You have to run this .exe in cmd prompt.

   **Command**: ada_line_count file1 file2 file3 …

2. **POSIX ADA Support (For Linux only) --** Ada tools now give support to POSIX. You have a separate library with POSIX Ada packages, for Linux only. To make this "linked library" available to a given user, the adaopts command:

   **Command**: adaopts -p /usr/local/OSPAL/Tools/Ada/linux/posix_adawill link the posix-ada library into the "search path" for the current library.

3. **Ada Support for GNAT compatibility compiler –** This feature enables you to link the "gnat compatibility" library into the search path for the currentlibrary. The following commands are used to link:

   **Linux Command**: adaopts -p /usr/local/OSPAL/Tools/Ada/linux/gnat_compat

   **Windows Command**: adaopts -p C:\OSPAL\Tools\Ada\windows\gnat_compat

4. **Ada Support for Win32 –** This feature enables support for Ada on Win32 host**.** The following command is used for the "win32ada" library**:**

   **Windows Command:** adaopts -p C:\OSPAL\Tools\Ada\windows\win32ada

**NOTE**: On Linux HOST/Environment, you need to set View/Modify permissions to the Tools folder.

**To set View/Modify permissions**:

- Go to the Tools folder.
- Right click on the Tools folder, and select **Properties**>**Permissions**.
- Change the required permissions.

Then provide executable permissions to files under [tools/Ada/linux/bin] folder before creating any Ada project. Otherwise it will give an OS PAL exception while trying to convert Ada to C using Ada-PAL Compiler options.

**To change executable permissions**:

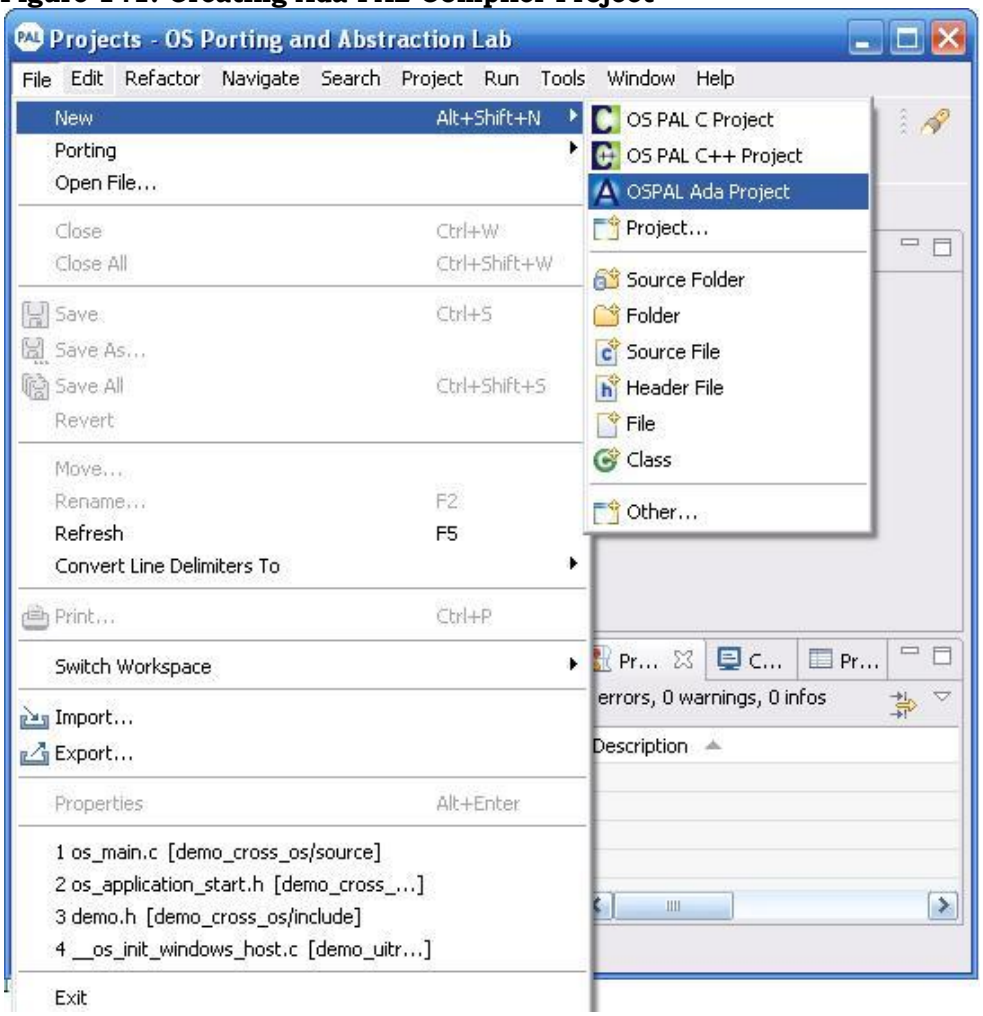
- Go to Terminal/Command window.
- Go to the respective folder location by cd OS-PAL root directory/Tools/Ada/linux/bin.
- Once you are in "bin" folder, run the command like <chmod 777 *>

Now observe files changes color from black to green.

**To create Ada-PAL Compiler project**:

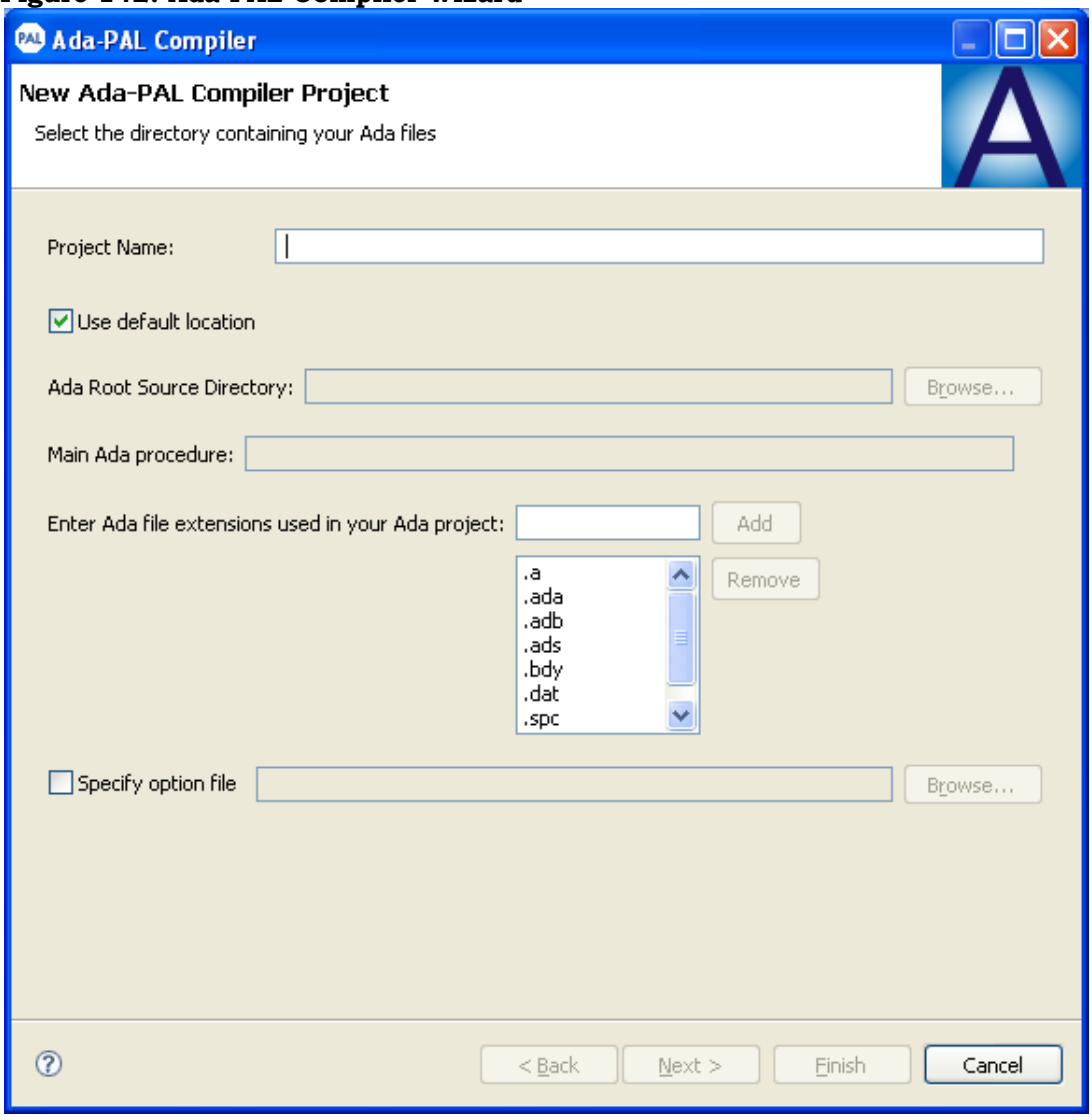
1. From OS PAL main menu, select **File > New >Ada-PAL CompilerProject**as shown in Figure 141, oron the projects pane, right click and select **Ada-PAL Compiler Project**.

**Figure 141: Creating Ada-PAL Compiler Project**



2. On Ada-PAL Compiler wizard, select the importing directory which consists of Ada sources that needs to be converted to C Sources, called as the Ada Program Library, by clicking on **Browse** button next to the text box as shown in Figure 142.

**Figure 142: Ada-PAL Compiler Wizard**



The field descriptions for Import Ada Files page are as follows:

| Field | Description | Your Action |
|---|---|---|
| Project Name | Specifies the user specific project name. | Enter a name to the project you want to create. |
| Use Default Location | Specifies the default location for project creation. | Enable the check box to use the default location for your project. If you want to specify another location, disable the check box, and browse to the necessary folder location.<br>**Note**: By default, this is enabled. |
| Ada Root Source Directory | Specifies the root directory which consists of Ada files with the specified extensions in the list.<br>**Note**: You can add multiple source directories in Ada-PAL Compiler Configuration page. | Select the root directory, by clicking **Browse**.<br>**Note**: You cannot enter anything in the root directory text box.<br>If you choose an empty directory, helloworld.ada will be copied into this directory. |
| Main Ada procedure | Specifies the main procedure name of the project the user imports that is converted to the main c function that will be started as thread in Cross-OS or other interfaces. | Enter the main Ada procedure name. For example: Sudoku_Test.<br>**Note**: The default value is hello_world |
| Enter Ada File Extensions used in your Ada Project | Specifies the source files that have extensions other than the default extensions listed in the drop down list. The default extensions listed here are: .a, .ada, .adb, .ads, .bdy, .dat, .spc, .sub | You can do any one of the following:<br>• To add a new extension other than the default ones, enter in the text box and click **Add**.<br>**Note**: By default, this is enabled.<br>• To remove an extension, select the extension from the drop down list and click **Remove**.<br>**Note 1:** Default extensions already available cannot be removed.<br>**Note 2**: Ada Extensions are case sensitive. |
| Specify Option File | Specifies if the user wants to specify any set of options that are needed for the Ada-C/C++Changer.<br>**Note**: This can be created using “space” as delimiter. | To specify an option file, select the check box and click **Browse** and select the option file.<br>**Note**: If option file is specified then Ada configuration options page will open with the specified options in the option file.<br>If not specified, then Ada Configuration options page opens with the default options. User can select or over ride the options in this page. |

# ADA-PAL Compiler Configuration Options

On the Ada-PAL Compiler Configuration Options page, you can set the following configurations:

- Ada Source
- Ada Listings
- Ada Messages
- Ada Drivers
- Additional

**NOTE**: You can change the configuration options on the Ada-PAL Compiler Property Page. To go to the Ada-PAL Compiler Property Page, right click on the project and select **Properties**.

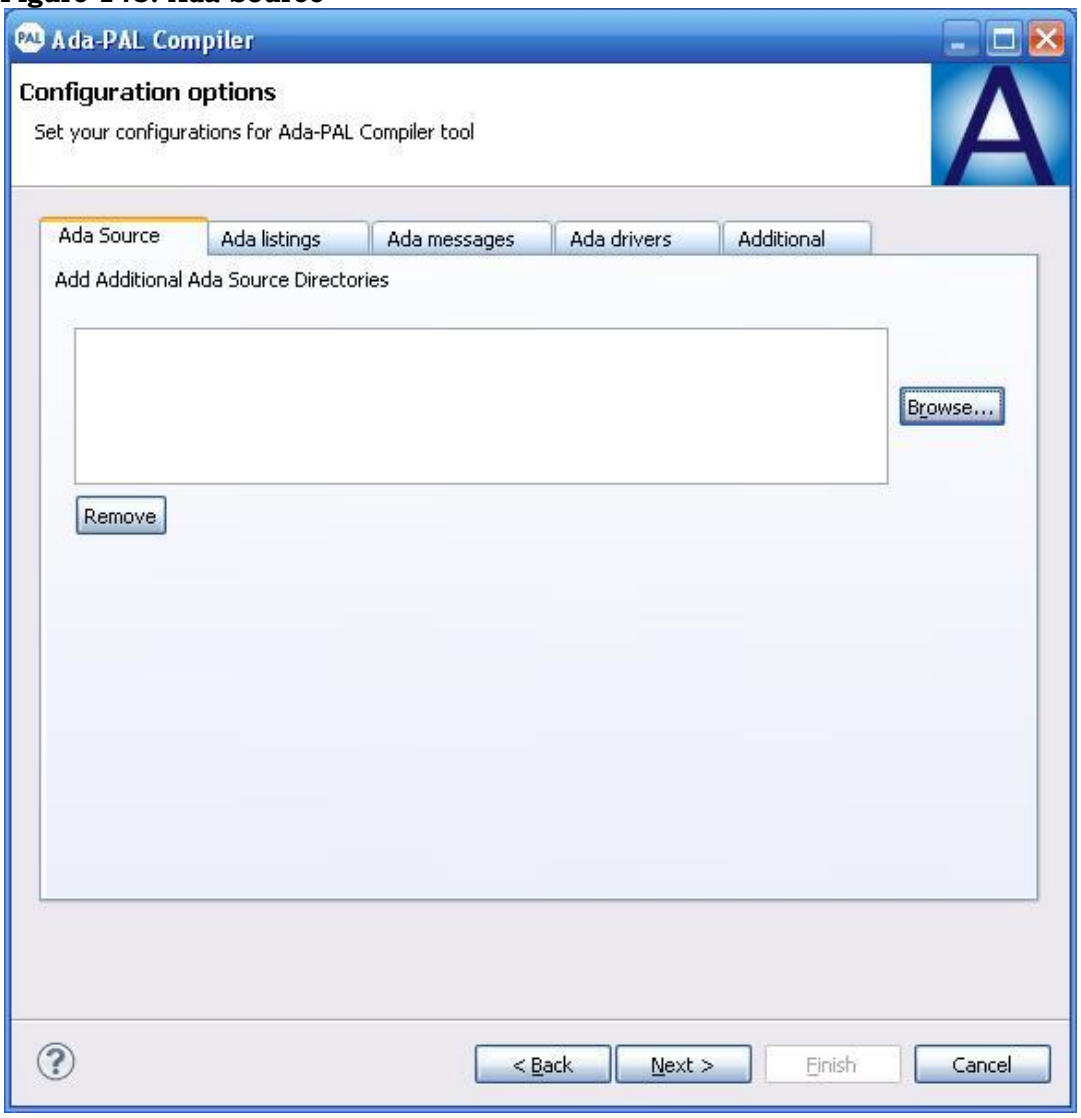
**Ada-PAL Compiler Configuration File:** Ada-PAL Compiler options are configurable via an Option's file.

a. Once you have anAda-PAL Compiler project and you want to use the same options that you have used.
b. Browse to the workspace directory location and into your project_name directory.
c. You will find a file called "options" file and you can save that file in a different location (you can also rename it if you want) and use it again and again.
d. You can create new Ada-PAL Compiler projects by passing the file info in the first screen. Ada-PAL Compiler reads this info and sets up the GUI configuration values accordingly.

   **Note**: (In the next release, we are thinking of skipping the entire subsequent configuration screen (until the OS Abstractor Integration screen).
e. This way you can create an option file and use it repeatedly as a template. However, if you later want to modify these options after creating the project, you can select the Ada-PAL Compiler project and right click and choose **Properties** and select **Ada-PAL Compiler Configurationpage** and change. This will get stored as the new option file for that project. This gives you the flexibility to use the template when you create the project and also let you change if needed.

3. OS PAL now provides support to multiple Ada source directories. You can browse and select the multiple source directories. On Ada Source tab page, add the Ada source directories as shown in Figure 143.

**Figure 143: Ada Source**

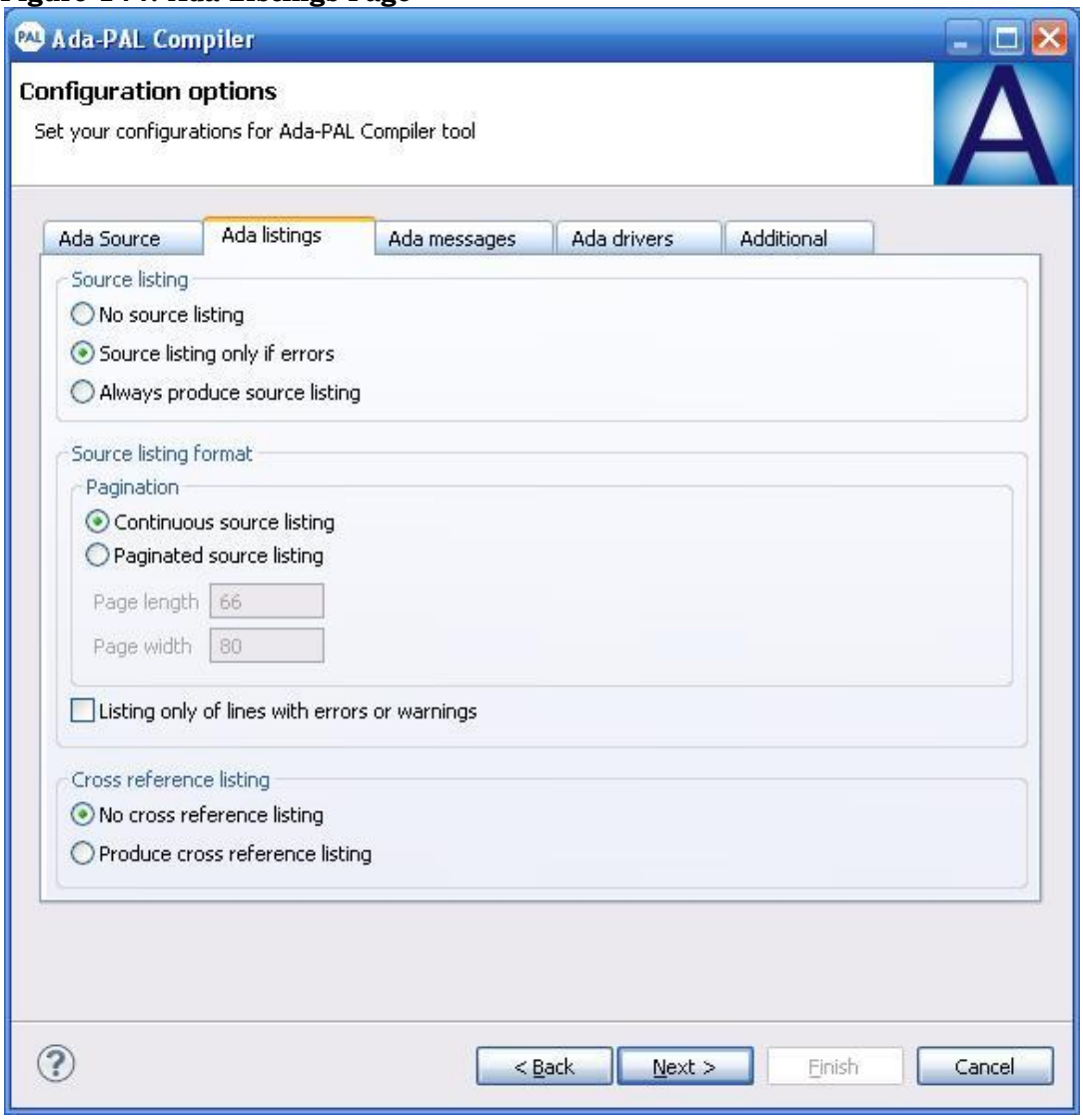


The field descriptions on Ada Source tab are as follows:

| Field | Description | Your Action |
|---|---|---|
| Add Additional Ada Source Directories | Specifies if you want to add any additional Source directories. | You can do any one of the following:<br>• To select the additional Ada – PAL source directories, click **Browse** and select the Ada source directories.<br>• You can also remove an Ada-PAL source directory by clicking **Remove**. |

4. On Ada Listings tab, set your Ada listings configurations as shown in Figure 144.

**Figure 144: Ada Listings Page**

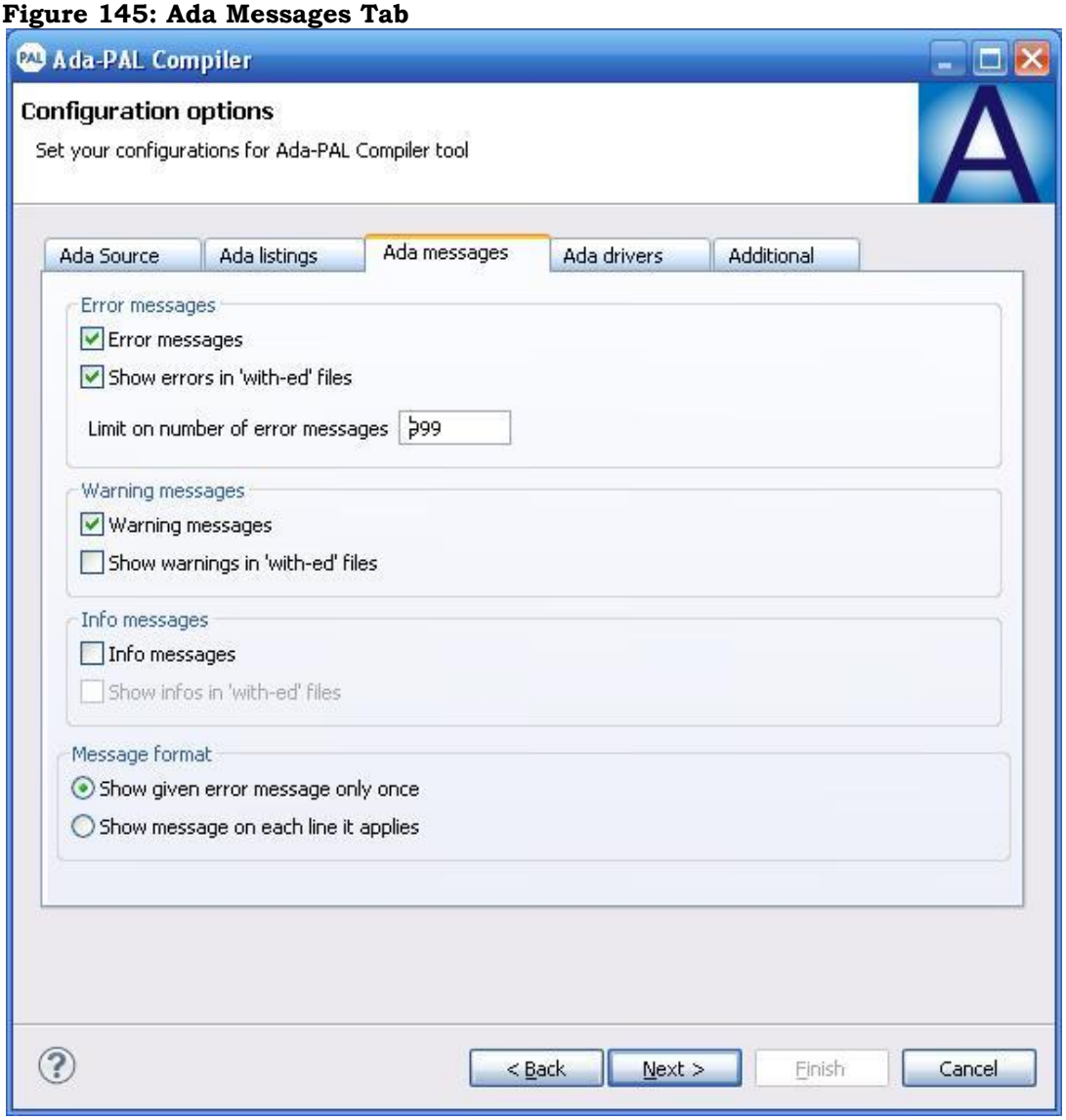


The field descriptions on Ada Listings tab are as follows:

| Field | Description | Your Action |
|---|---|---|
| Source Listing | Specifies if you want the Ada source list to be generated or not and how it should be generated. | You can do any one of the following:<br>• To not to do Ada source list, select the radio button.<br>• To do Ada source list only if errors are present, select the radio button.<br>**Note**: By default, this is enabled.<br>• To always produce Ada source list, select the radio button. |
| Source Listing Format | | |
| Pagination | Specifies the format of the Source Listing. | You can do any one of the following:<br>• For continuous source listing, select the radio button.<br>**Note**: By default, this is enabled.<br>• For listing only of lines with errors or warnings, select the radio button.<br>• For paginated source listing, select the radio button. |
| Page Length | Specifies the length of the page. | Enter a value for the length of the page.<br>**Note**: By default, the value is 66. |
| Page Width | Specifies the width of the page. | Enter a value for the width of the page.<br>**Note**: By default, the value is 80. |
| Listing only of lines with errors or warnings | Specifies if you want to list only lines with errors or warnings | Select the check box. |
| Cross Reference Listing | | |
| No cross reference listing | Specifies if you want to generate a cross reference listing. | To not to generate a cross reference listing, select the radio button.<br>**Note:** By default, this is enabled. |
| Produce cross reference listing | Specifies if you want to generate a cross reference listing. | To generate a cross reference listing, select the radio button.<br>**Note:** By default, this is disabled. |

5. On Ada Messages tab, describe Ada message options as shown in Figure 145.

**Figure 145: Ada Messages Tab**

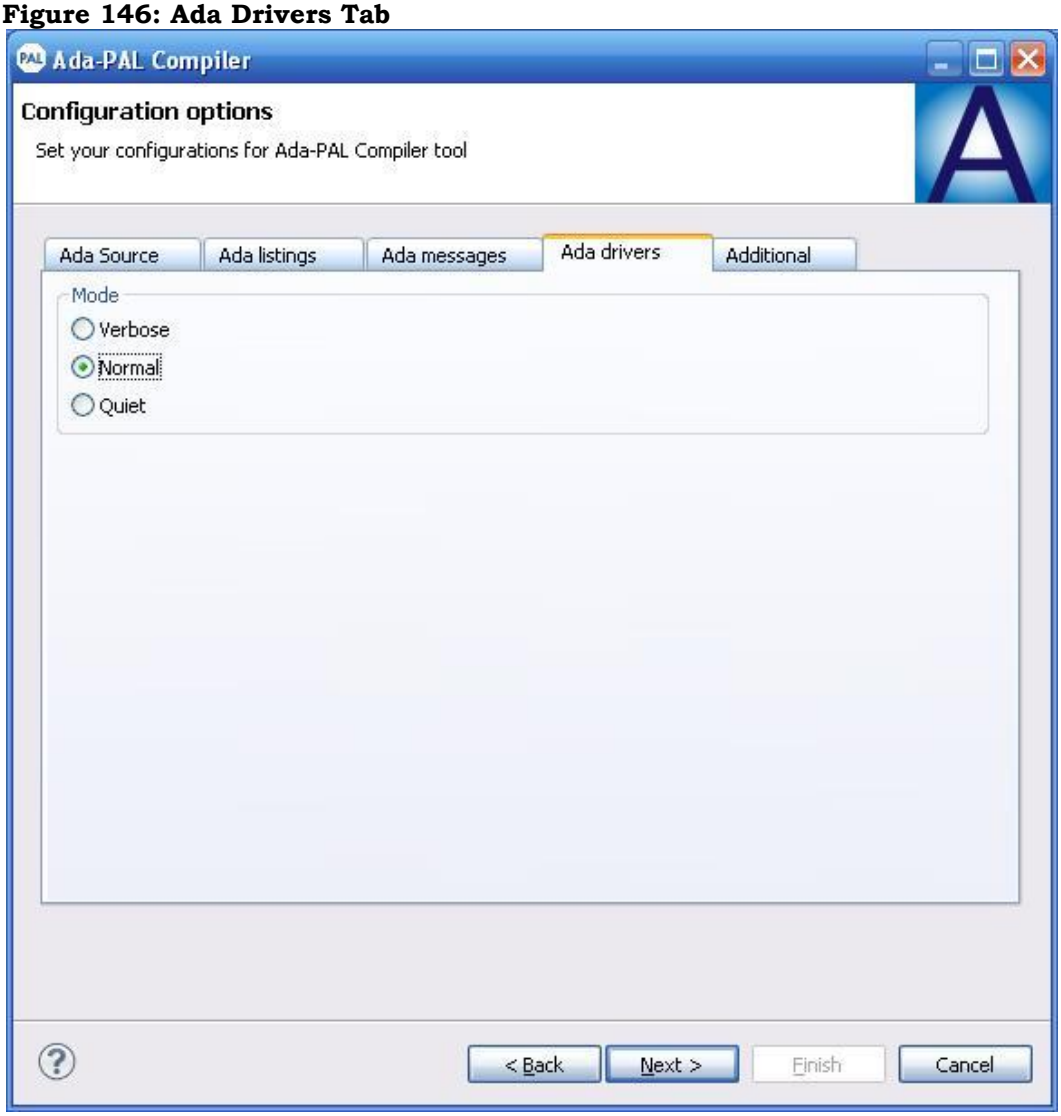


The field descriptions on Ada Messages tab are as follows:

| Field | Description | Your Action |
|---|---|---|
| Error Messages | Specifies if you want to generate error messages. | You can do any one of the following:<br>• To select error messages, select the check box.<br>**Note**: By default, this is enabled.<br>• To show errors in 'with-ed' files, select the check box. |
| Limit on number of error messages | Specifies the limit on the count of error messages. | Enter a value to specify the limit on number of error messages.<br>**Note**: By default, the value is 999. |
| Warning Messages | Specifies if you want to select warning messages. | You can do any one of the following:<br>• To select warning messages, select the check box.<br>**Note**: By default, this is enabled.<br>• To show warnings in 'with-ed' files, select the check box. |
| Info Messages | Specifies if you want to select the info messages. | You can do any one of the following:<br>• To select info messages, select the check box.<br>• To show infos in 'with-ed' files, select the check box.<br>**Note**: By default, this option is disabled. |
| Message Format | Specifies the format of the message. | You can do any one of the following:<br>• To show given error message only once, select the radio button.<br>**Note**: By default, this is enabled.<br>• To show message on each line it applies, select the radio button. |

6. On Ada Drivers tab, set Ada driversconfigurations as shown in Figure 146.

**Figure 146: Ada Drivers Tab**

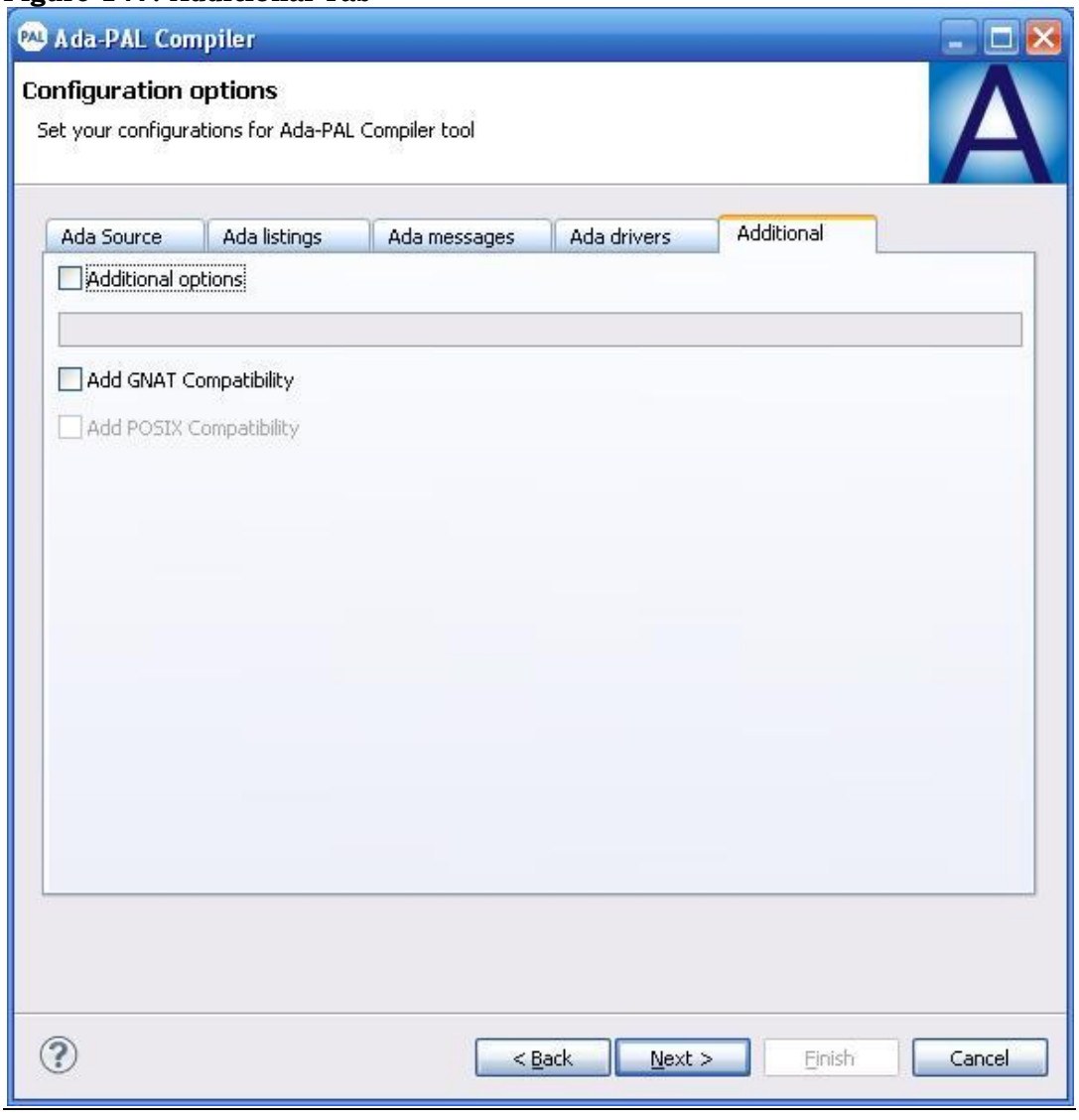


The field descriptions for Ada Drivers tab are as follows:

| Field | Description | Your Action |
|---|---|---|
| Mode | Specifies the required mode for reporting the compiler actions. | You can do any one of the following:<br>• To select the verbose mode, select the radio button.<br>• To select the normal mode, select the radio button.<br>**Note**: By default, this is enabled.<br>• To select the quiet mode, select the radio button. |

7. On the Additional tab, set additional configurations for Ada-PAL Compiler and click **Next** as shown in Figure 147.

**Figure 147: Additional Tab**

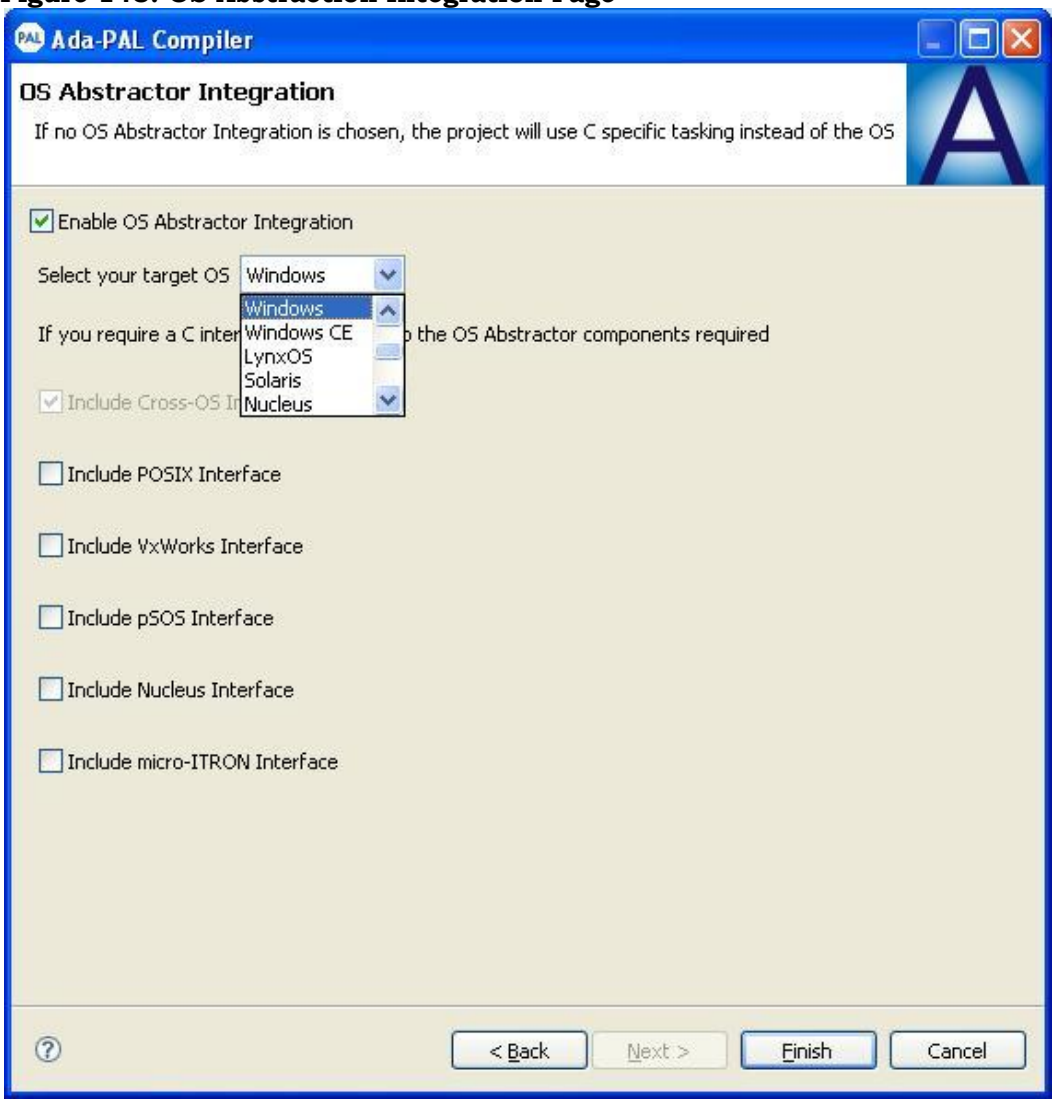


The field descriptions for Additional options tab are as follows:

| Field | Description | Your Action |
|---|---|---|
| Additional Options | Specifies if you want to include any other additional options such as custom or optional. | To specify additional options, select the check box and enter a value in the text box. |
| Add GNAT Compatibility | Specifies if you want to add GNAT Compatibility. | To add GNAT compatibility, select the check box. |
| Add POSIX Compatibility | Specifies if you want to add POSIX Compatibility.<br>**Note**: This feature is not supported on Windows. It is supported on Linux only. | **Note**: This feature is disabled on Windows.<br>On Linux, to add POSIX compatibility, select the check box. |

8. On OS Abstractor Integration page, select the check box to enable OS Abstractor Integration.And select your target OS by selecting from the drop down list of available Ossas shown in Figure 148.
   **NOTE 1**: If no OS Abstractor Integration is chosen, the project will use C specific tasking instead of the OS.
9. If you require a C interface, select the check box next to the required OS Abstractor component, and click **Finish** as shown in Figure 148.
   **NOTE 2:** If you have enabled the Cross-OS APIs, you can enable the additional development APIs after importing to OS PAL.

**Figure 148: OS Abstraction Integration Page**



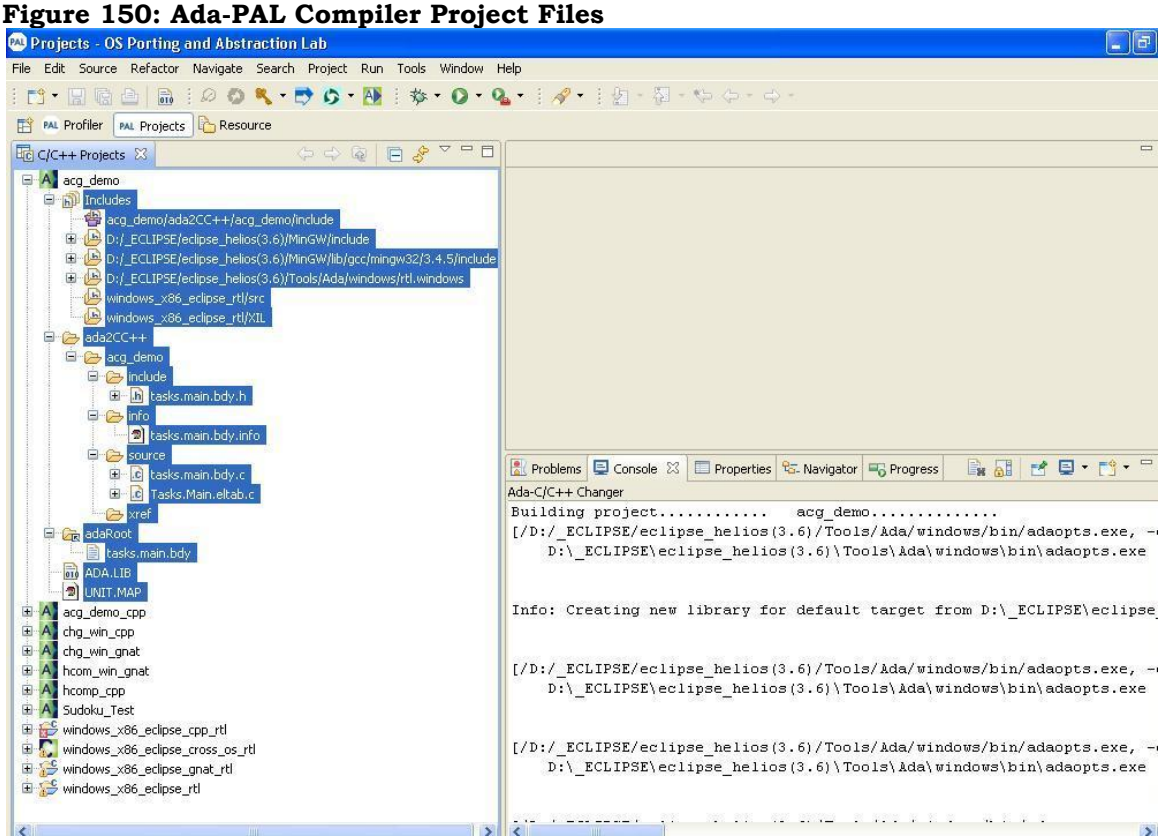
10. After compiling the Ada code to executable, file generation is successfully done.The porting report page is displayed as shown in Figure 149. Click **Done** to complete the process.

**Figure 149: Ada-PAL Compiler Porting Report**


You have successfully converted Ada sources to object or binary executables.

11. You can view the created Ada-PALCompiler projectunder projectspane as shown in the Figure 150**.**

**Figure 150: Ada-PAL Compiler Project Files**



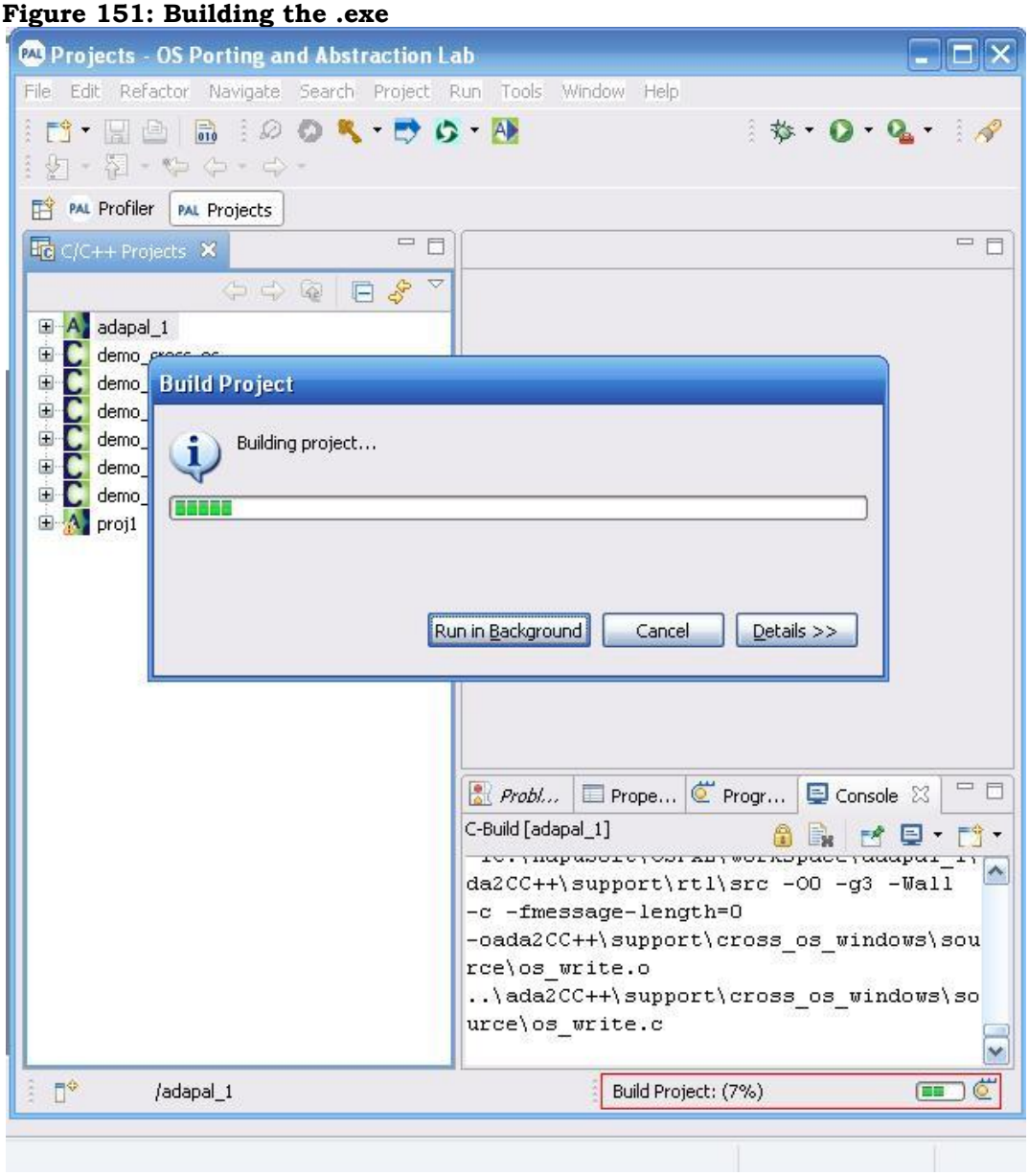
The Ada-PAL Compiler project has the following files:

- **Includes**–This folder contains the header file paths of your project
- **Ada2CC++**–This folder contains all the files generated by Ada tools
    - **Ada-PAL Proj1**–This folder contains the Ada-PAL Compiler project related files
        - **Include**–This folder contains the Ada header files.
        - **info–**This folder contains information file subdirectory of the program library directory where information files for the object modules are placed.
        - **source**–This folder contains the converted C source files.
        - **init**–If OS Abstractor is integrated, then you get a folder, which contains the OSPAL template files. **NOTE:** For more information on the Template files, refer to OS PAL Template Files section in this manual.
        - **xref–**This folder contains the cross-referenced files which are generated by the Ada tools.
- **adaRoot–** This folder contains the Ada sources added during your project creation. In case, you add any additional sources, you can view this in the **Project > Property page > Ada Source** tab of the respective project.
- **ADA.LIB–This** contains information describing the configuration of the Ada library
- **UNIT.MAP**–This contains a unit-to-source mapping for use by the compiler and program builder
- **.options**–This contains the list of options with which Ada-PAL Compiler project or executable is created. This is a hidden file. You can view this in Navigator view. To view, select **Window > Show View > Other > General > Navigator**.

**NOTE**: Host Libraries and include paths are automatically added during project creation. For viewing this information, select **Ada-PAL Compiler Project > Properties > C/C++ Build > Settings**.

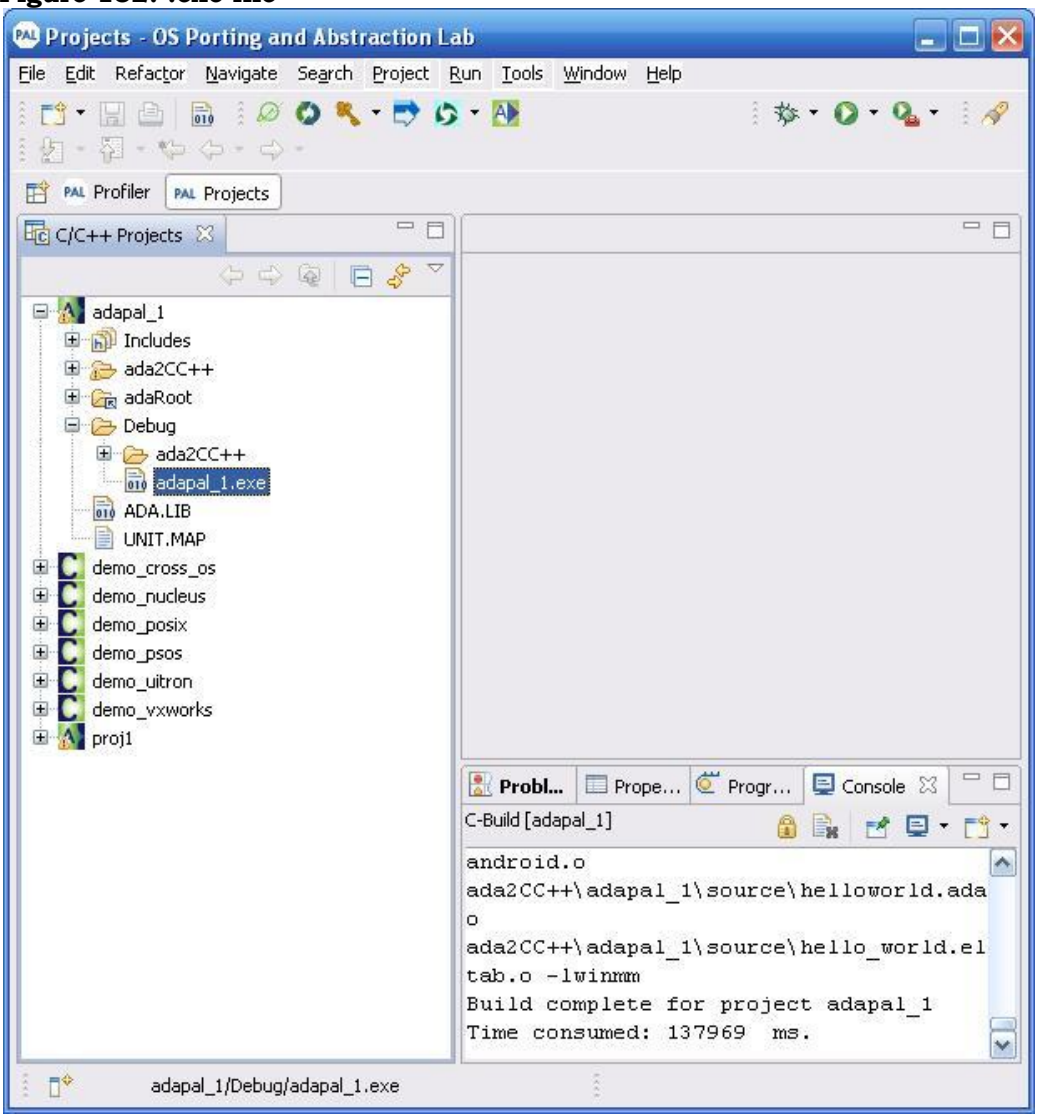
12. To generate the .exe file, select the Ada-PAL project you have created and right click and select **Build Project** as shown in Figure 151.

**Figure 151: Building the .exe**



13. To view the .exe file generated, expand the project you have created and expand the **Debug** folder. You can view the .exe file generated by the Ada-PAL Compiler as shown in Figure 152.

**Figure 152: .exe file**




**NOTE:** While importing an Ada project, you will receive 3617 warning messages. It appears that most, if not all of them, are associated with the rtl files.

They are as follows:

- Defined but not used variables(3129)
- Return with no value, in function returning non-void (15)
- Assignment from incompatible pointer type (52)
- Comparison is always false due to limited range of data type (94)
- Cast from pointer to integer of different size (3)
- Comparison of distinct pointer types lacks a cast (8)

- Control reaches end of non-void function (71)
- Implicit declaration of function (10)
- Initialization from incompatible pointer type (5)
- Integer constant is so large that it is unsigned (1)
- Left shift count >= width type (1)
- Missing braces around initializer (2)
- Passing arg from incompatible pointer type (114)
- Statement with no effect (28)
- Unused variable (83)
- This decimal constant is unsigned only in ISO C90 (1)

## Ada-PAL Compiler Build

Ada-PAL Compiler enables you to build an existing project. This feature enables you to either do an incremental build or a full build on your Ada 95 sources.

- The Build process will incrementally compile the Ada files that have been modified or added since the last build.

  **NOTE**: To do an incremental build, you should not do **Clean**.

- If any new Ada source files are added, removed, or modified in the project, and want to generate the c- sources again, you can do a full build by first calling **Clean** and then **Build.**

  **NOTE: Clean** will delete all your info files, which will result in a full build.

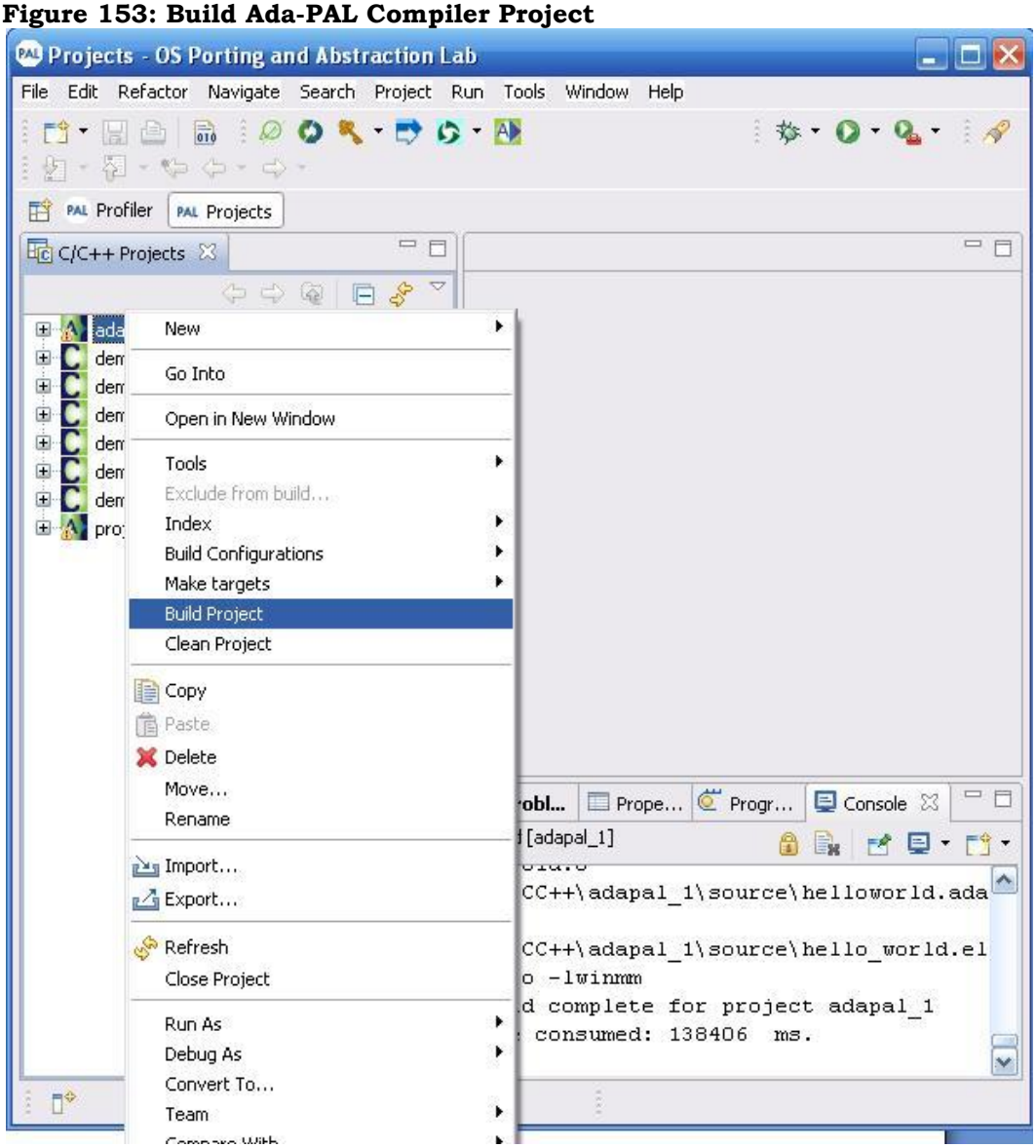
- You cannot re-build if new directories are created or new files are added with differing extension than what was provided during the project creation. If you have new directories and new extensions, then you must recreate the Ada-PAL Compiler project.

**Ada-PAL Compiler project Build**

**To build an Ada-PALCompiler project**:

1. Create an Ada-PALCompiler project using the Ada-PAL Compiler. Refer to Creating Ada-PAL Compiler project section.
2. Select the project and right click on it and select **Build** as shown in the Figure 153.

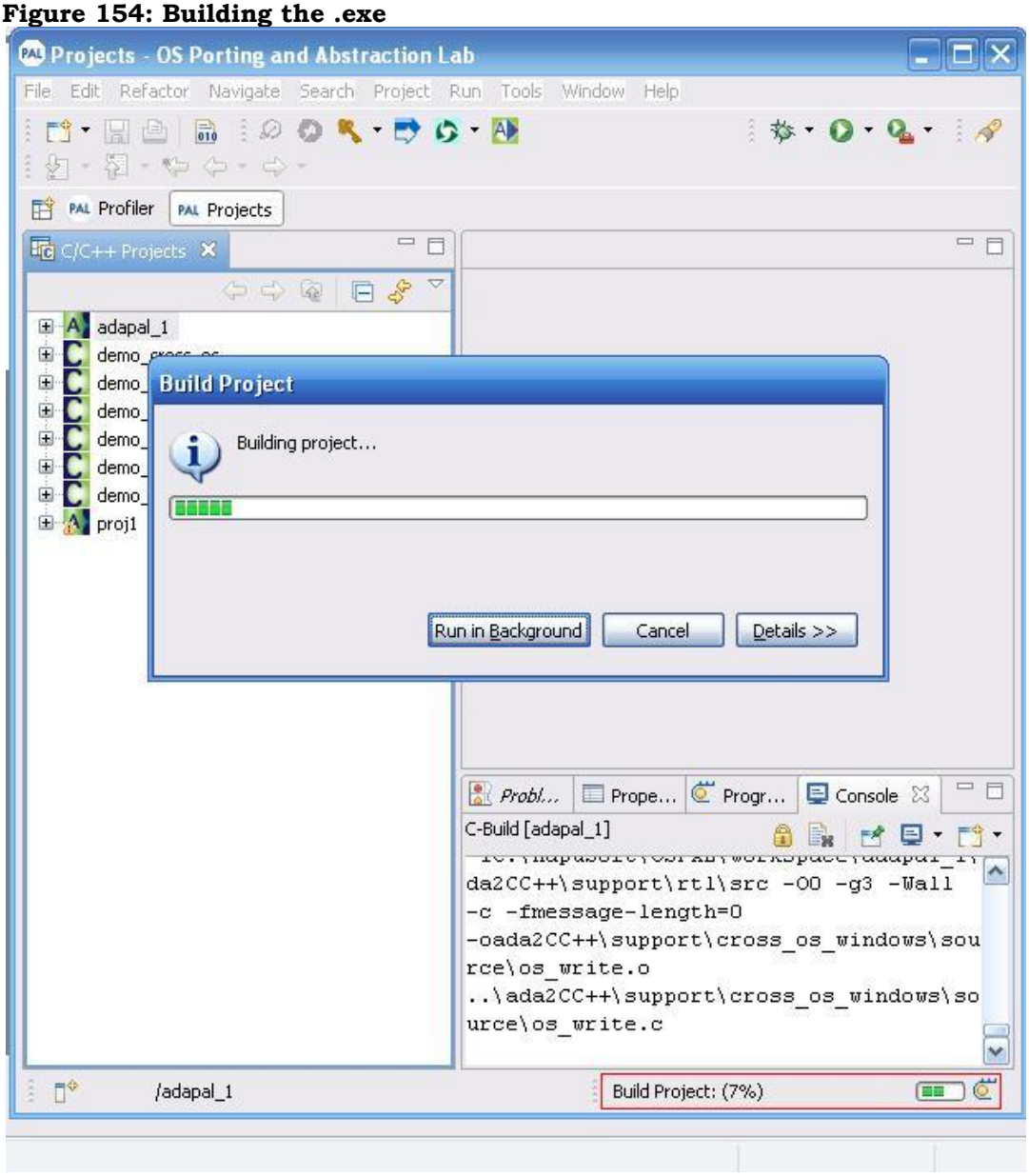
**Figure 153: Build Ada-PAL Compiler Project**




You can view the changes or modifications, if any, in adaRoot folder.

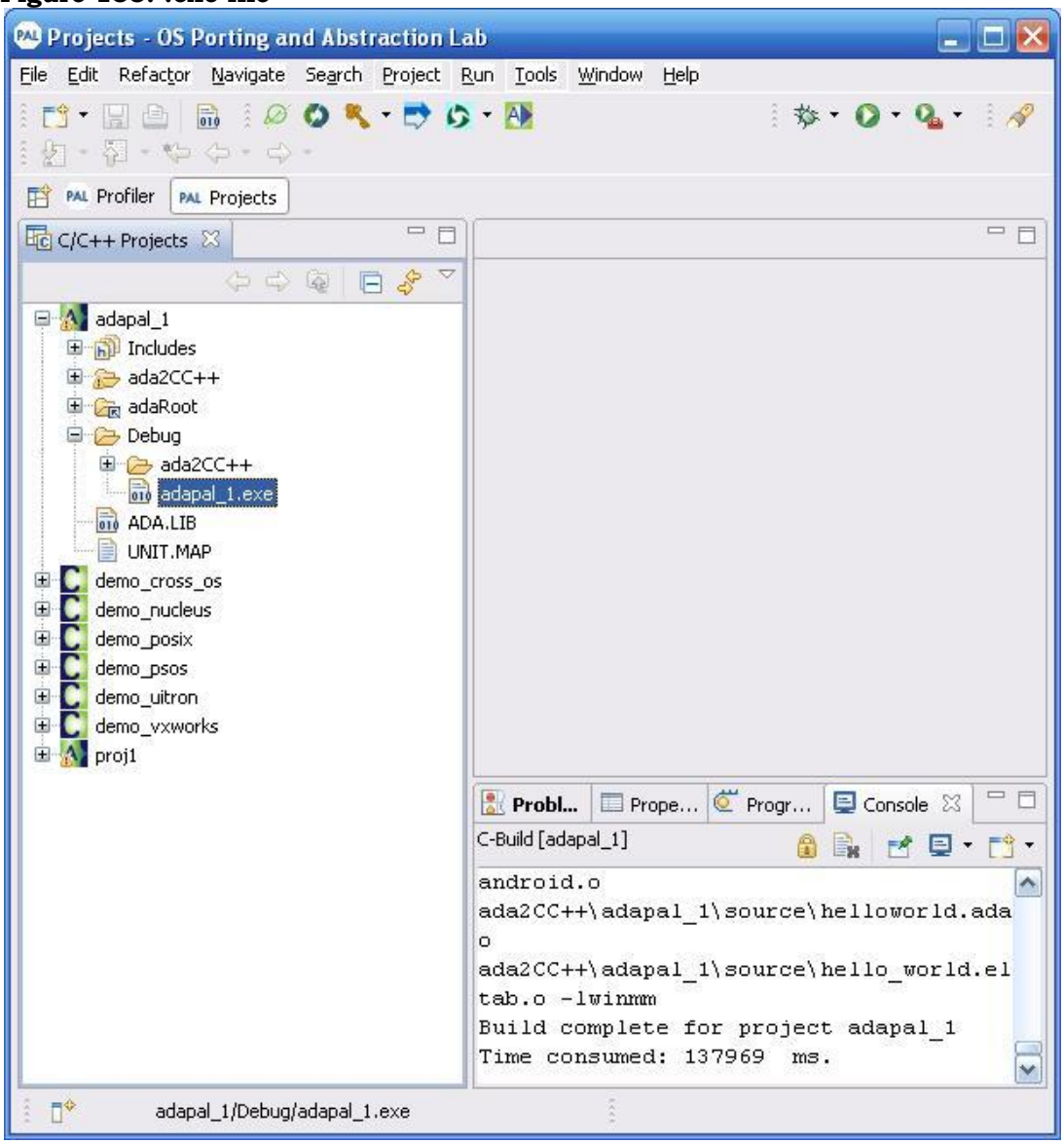
14. To generate the .exe file, select the Ada-PAL project you have created and right click and select **Build Project** as shown in Figure 154.

**Figure 154: Building the .exe**



15. To view the .exe file generated, expand the project you have created and expand the **Debug** folder. You can view the .exe file generated by the Ada-PAL Compiler as shown in Figure 155.

**Figure 155: .exe file**




**NOTE:** While importing an Ada project, you will receive 3617 warning messages. It appears that most, if not all of them, are associated with the rtl files.

They are as follows:

- Defined but not used variables(3129)
- Return with no value, in function returning non-void (15)
- Assignment from incompatible pointer type (52)
- Comparison is always false due to limited range of data type (94)
- Cast from pointer to integer of different size (3)

- Comparison of distinct pointer types lacks a cast (8)
- Control reaches end of non-void function (71)
- Implicit declaration of function (10)
- Initialization from incompatible pointer type (5)
- Integer constant is so large that it is unsigned (1)
- Left shift count >= width type (1)
- Missing braces around initializer (2)
- Passing arg from incompatible pointer type (114)
- Statement with no effect (28)
- Unused variable (83)
- This decimal constant is unsigned only in ISO C90 (1)

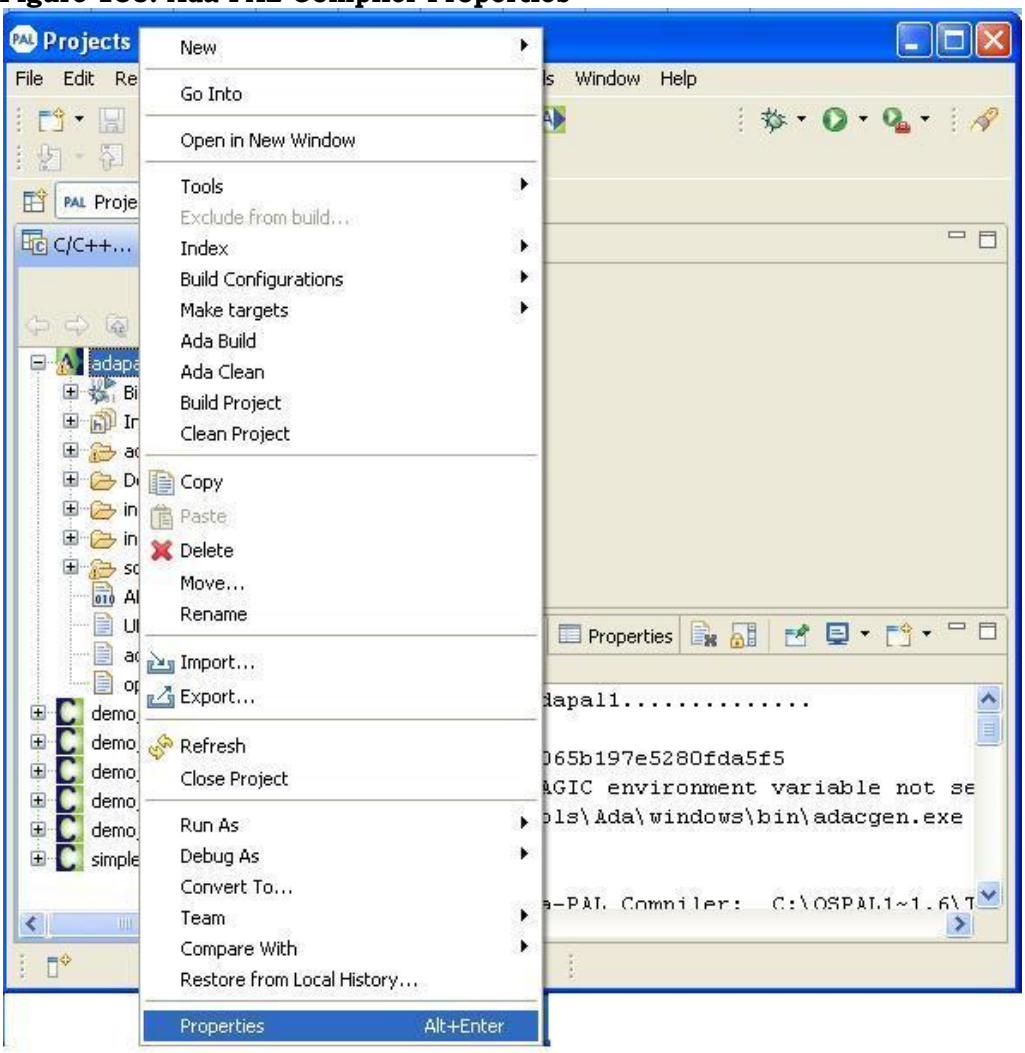
## Ada-PAL Compiler Property Page

Ada-PAL Compiler Property Page enables you to change or modify the configuration options you have set for your project.

To go to the property page:

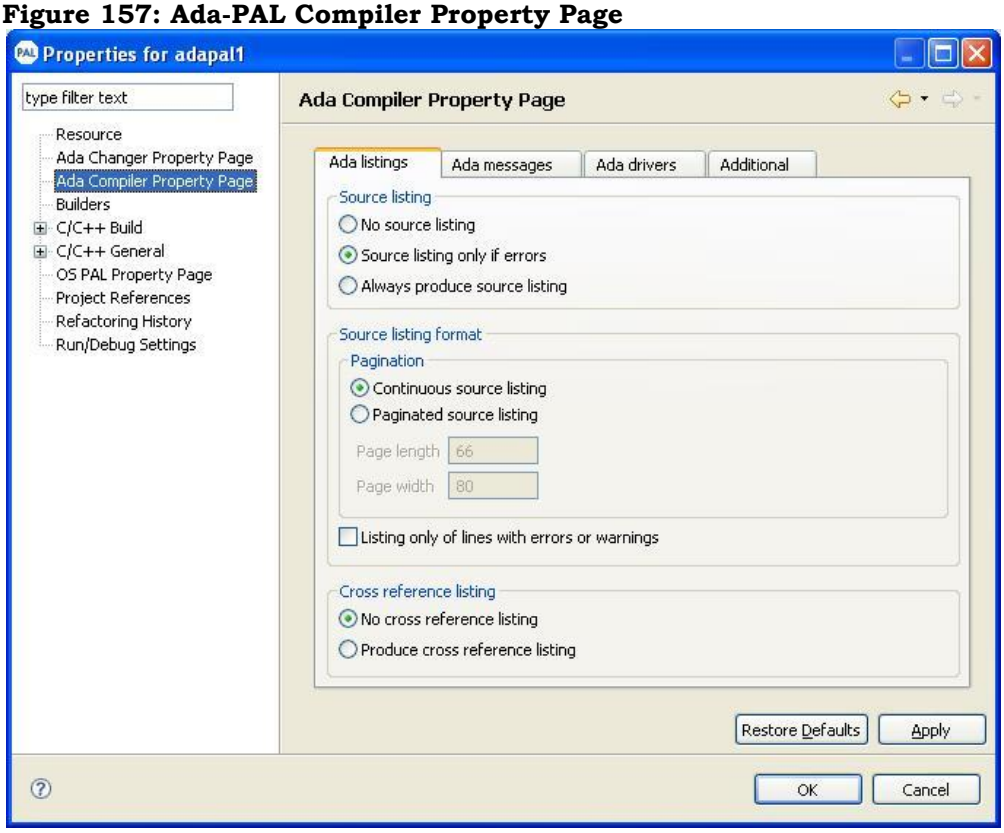
1. On OS PAL Projects pane, select the Ada-PAL Compiler project you have created. Right click on it and select **Properties** and shown in the Figure 156.

**Figure 156: Ada-PAL Compiler Properties**



2. The Ada-PALCompiler property page is displayed as shown in the Figure 157. Make the necessary changes and click **Apply**.
3. To change the Main Procedure Name, on the Ada-PAL property page, click on **Additional** tab, and make the necessary changes and click **Apply**.

**Figure 157: Ada-PAL Compiler Property Page**




For more information about the Ada Configuration Options, refer to ADA-C/C++Changer Configuration Options section.

When you do an Ada Build, you can re-import files, or import deleted files, remove any files or change the Main procedure name on this page. You can modify on any of the configurations on the property page.

**NOTE**: You cannot edit or modify the AdaRoot Directory location.

## Target Code Generation On Ada-PAL Compiler Project

For Ada-PAL Compiler Projects, Cross-OS interfaces are added directly to the project as targets sources if the user has a valid and relevant Standalone Package Generator license. If Target Code Generation is attempted on these projects, all the Cross-OS functionality, being part of application, is again redefined in cross_os.c. This will give re-definition errors on compile time.

Hence Ada-PAL Compiler projects cannot be code-optimized.

## Manual Modifications to Projects files generated by Target Code Generator

The target code output produced when optimizing Ada projects via the target code optimization process is a little different than that of the standard C/C++ OS PAL project. In this case, the API level optimization process is skipped as the application needs to link-in other required RTL C/C++ libraries and possible other 'C' interface libraries. Instead of the OS Abstractor code being included as part of the application, it is added into the target directory as a separate code base that should be built as libraries. There will be separate libraries for the Cross OS component as well as any other OS Interface components (like VxWorks, Windows, etc.) included in the project. There will also be a separate Ada Run Time Library(RTL) required to be linked in as well.

For example, if a converted Ada to C/C++ project that was integrated with OS Abstractor and includes the POSIX Interface were optimized for a windows target it would look like follows:

```
<target dir>
    cross_os_windows
        source
        include
        specific
    posix_interface
        source
        include
        specific
    include
        include
    rtl
        XIL
        src
        IL
    <app name>
        ada2C++
            <app name>
                source
                include
```

The <target_dir> is the directory location where the generated code would be placed. The Cross OS and OS Interface directories will include project files specific to your target. Project files for the RTL will only be included for Windows and Linux targets. On a Windows target, you will get a project file for the Eclipse IDE and on Linux you will get both Eclipse and make files. For all other targets and toolsets you will need to

create an RTL library project. An application project will be created for the target, but it will require some manual modifications to build.

The modifications which need to be made to the application project are as follows:

**Header Inclusion**

```
    Add include paths for the Ada RTL component.
        <target dir>/rtl
        <target dir>/rtl/src
        <target dir>/rtl/IL

    Add include path for any other 'C' library that you need for your
application

    if your Ada project is integrated with Cross OS you will need to
add the following:
        <target dir>/include/include
        <target dir>/cross_os_<target>/include
```

# Revision History

**Document Title: OS Porting and Abstraction Lab User Manual in MS Word**
**Release Number: 1.3.7**

| Release | Revision | Orig. of Change | Description of Change |
|---|---|---|---|
| 1.3.6 | 0.1 | VV | New Manual |
| 1.3.6.1 | 0.1 | VV | Ada sections |
| 1.3.7 | 0.1 | VV | Changes to Ada-C/C++ Changer and Target Code Optimization sections |
| 1.3.8 | 0.1 | VV | Changes to Target Code Optimization sections |